This chapter describes the AOCE Standard Mail Package. The AOCE Standard Mail Package provides a high-level interface that makes it easy for you to add electronic-mail capabilities to your applications.

The Standard Mail Package provides two separate services:

- an easy way to send a letter or a file from within your application without user intervention
- a complete user interface that you can use to convert any of your application's documents into electronic mail

In addition, you can use the Standard Catalog Package to provide a user interface for browsing AOCE catalogs and selecting records from within your application.

If you want to design and implement your own electronic messaging service using the AOCE toolbox, see the chapter "Interprogram Messaging Manager" in this book.

About the Standard Mail Package

The AOCE Standard Mail Package provides a high-level interface to the AOCE Interprogram Messaging (IPM) Manager. It works together with the Catalog Browser and the Digital Signature Manager to present a consistent and easy-to-use user interface for addressing letters, signing letters, and sending your application's documents as electronic mail.

The Standard Mail Package can be divided into two main parts: the send-letter functions and the mailer functions. The Standard Mail Package relies on other components of the Apple Open Collaboration Environment, but you do not have to call the underlying AOCE services directly to add electronic-mail capabilities to your application.

The Send-Letter Functions

The Standard Mail Package provides a basic, very easily implemented method of sending documents and other files that can be used either by users of applications or by applications acting without user intervention (agents). You can use the Standard Mail Package functions (described in "Send-Letter Functions" on page 3-37) to enclose a file with an AppleMail letter and then send the letter, to send a document as an image file, or to send a file so that it appears in the recipient's In Tray not as a letter but as the original file.

The send-letter functions provide no interface for opening a letter from within your application. When the user double-clicks a document in the In Tray, the Finder attempts to launch the application that was used to send the letter, and that application opens the document. If that application is not present, the Finder displays a dialog box asking the user whether it should open the letter with the AppleMail application provided with the AOCE software.

The Mailer Functions

The Standard Mail Package also provides a more sophisticated electronic-mail interface. This interface adds a new region—known as a ***mailer***—to any window.

"Providing Mailers in Your Windows" on page 3-45 describes functions to create a new mailer, reposition a mailer in your window, control the way the user cycles through fields in the mailer and your document using the Tab key, and dispose of a mailer.

You can use the functions described in "Handling Events in Mailers" beginning on page 3-63 to handle events and Apple events that pertain to mailers and to make sure that your menu commands accurately reflect the state of the mailer while a user is working in it.

You use the functions described in "Sending and Saving Mail" beginning on page 3-72 to send, save, or read a document containing a mailer. Use the routines in "Printing Mailers" beginning on page 3-107 when you want to print a document containing a mailer.

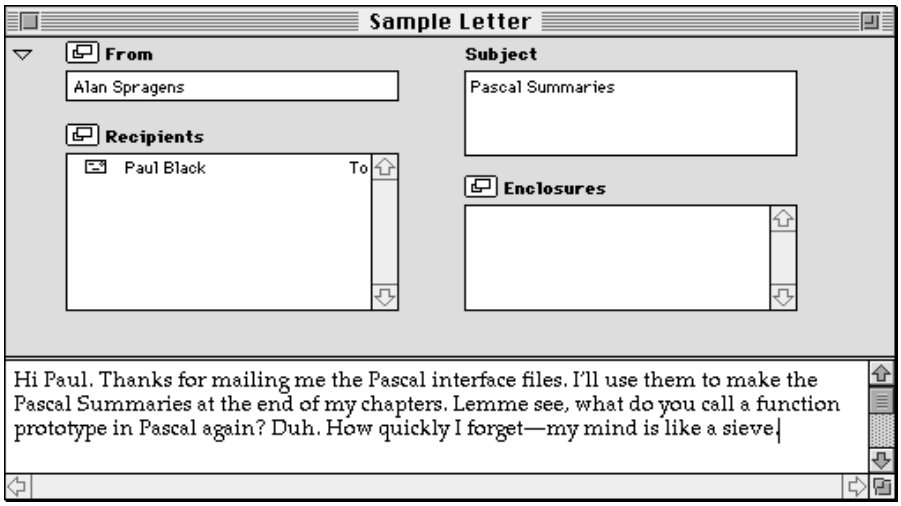
You can use the functions described in "Getting and Setting Information in the Mailer" beginning on page 3-110 to read and set values in mailer fields and send options from within your program instead of through the mailer or the standard dialog boxes provided by the mailer.

You use the `SMPOpenLetter` function, described in "Reading Mail" beginning on page 3-93, to open a letter to read its contents.

Mailers

The mailer lets the sender enter addresses and subject information, enclose other files and folders in the letter, and add a digital signature to the letter. It lets the recipient read all of this information and verify the digital signature. Figure 3-1 shows a mailer in an application window. Each time the user forwards a letter, another mailer holding addresses for the forwarder and the new recipients is added to the letter. The mailers for a forwarded letter are collectively referred to as a ***mailer set.***

Figure 3-1 Mailer in an application window



The preferred user interface for an AOCE Standard Mail Package letter is to place the mailer inside your document's windows, just below the title bar. However, if your application's windows are not suitable for displaying mailers, you can place the mailer in its own, separate window. The user can display the mailer in either of two states: *contracted* or *expanded*. Figure 3-2 shows a sample mailer in the contracted state and Figure 3-3 shows the same mailer in the expanded state.

Figure 3-2 Mailer in the contracted state

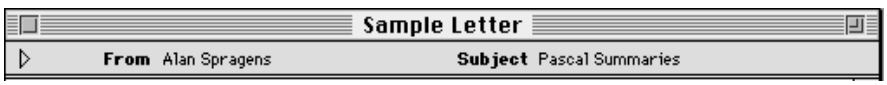
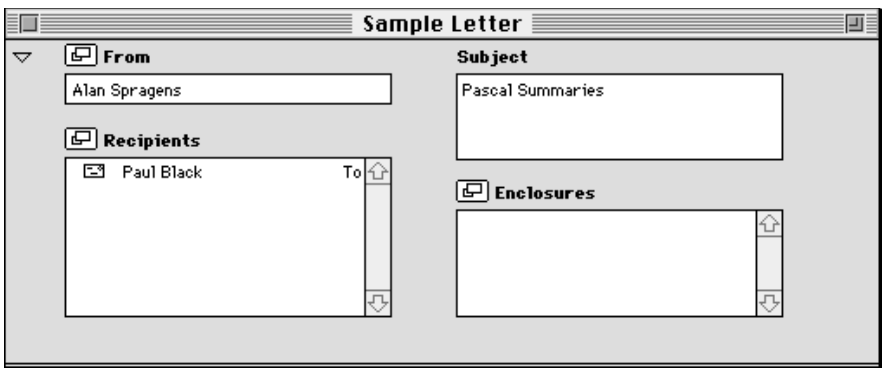


Figure 3-3 Mailer in the expanded state



The user can drag an address from the Finder or another mailer into the Recipients field or can open an addressing panel, as shown in Figure 3-4. The user can select among four versions of the addressing panel by clicking one of the icons at the left side of the panel. These versions of the addressing panel allow the user to select an address from the default personal catalog or from any AOCE catalog, to find a record by typing in all or part of the name of the record, or to type in the entire address. These four versions of the panel are shown in Figure 3-5.

Figure 3-4 Mailer with addressing panel open

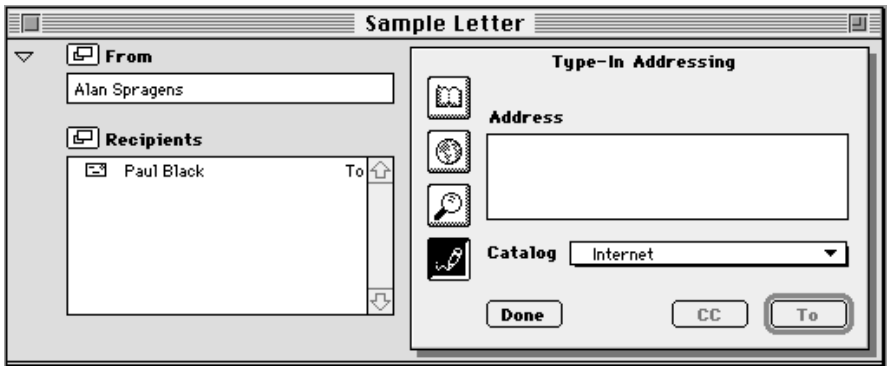
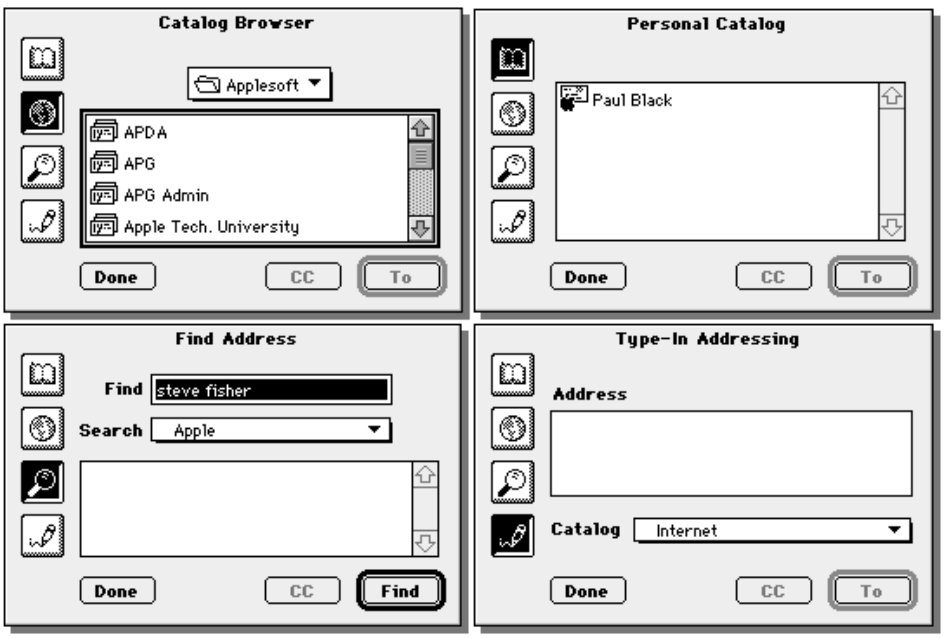


Figure 3-5 The four versions of the addressing panel



Letter Formats

When you use a mailer to send a document as a letter, you can send the document in a "native" format (that is, any one of the document formats supported by your application), you can send an image of your document, you can send the content of your document in a special format called ***standard interchange format,*** or you can send the document in any two or all three of these formats simultaneously.

You may choose to employ the AOCE Standard Mail Package at either of two levels: full mailer support or basic mailer support. An application that offers full mailer support can read and write both standard interchange format and images. An application that offers basic mailer support can send either images or standard interchange format or both. Either type of application might also send documents in one of the application's native formats.

When you send your document, the Standard Mail Package delivers it to the addressees' In Trays. When a recipient double-clicks a document in the In Tray, the application used to send the document (if present) opens it, and the mailer appears at the top of the window. If the file includes an image of the document or a standard interchange format version of its content, any application that offers full mailer support can open it.

Each user who has AOCE software has the AppleMail application, which provides full mailer support. Thus, every user who has AOCE software can read, either as an image or in standard interchange format, every document sent by an application that provides either full or basic mailer support. In addition, if your application can send and read documents sent in its own native formats, users who have your application have access to the complete document when they receive it.

A letter consists of a header that contains addressing and priority information, followed by blocks of data, followed by enclosures. Certain types of data blocks have standard definitions, such as the image block and standard interchange format blocks. The image block contains an image of the document being sent; you must provide an image-drawing routine (page 3-123) to draw each page. The `SMPImage` function (page 3-88) creates the image block and adds it to the letter. Standard interchange format blocks contain a version of your document that can include text, styled text, sounds, pictures, and QuickTime movies. Standard interchange format can be converted by access modules and read by any standard letter application (such as the AppleMail application provided with the AOCE software). You can define other blocks in any way you wish. You use the `SMPAddBlock` function (page 3-91) to add blocks to a letter.

You can send your own document in one of its native formats as an enclosure to the letter, known as a ***main enclosure*** (also referred to as a *content enclosure*), or incorporate it into data blocks, as you wish. The Standard Mail Package user interface also allows the user to enclose other files. (The main enclosure is not visible to the user as an enclosure.)

Note
If you are using the IPM Manager or the MSAM API to send letters to the Standard Mail Package, you should avoid sending any nested letters that contain standard content. If the Standard Mail Package receives a letter that contains a nested letter, it ignores any content (standard interchange format or image format) within the nested letter. It displays the header and nesting information of the nested letter as a forwarded mailer. ◆

The Standard Catalog Package

The Standard Catalog Package provides authentication and letter-addressing services that complement the routines described in this chapter. See the chapter "Standard Catalog Package" in this book for more information.

Using the Standard Mail Package

This section describes how to initialize the Standard Mail Package and use it to create a mailer, send mail, receive mail, forward and reply to mail, close a letter, and handle events in the mailer.

Initializing the Standard Mail Package

Before you can enable Standard Mail Package features in your application, you must use the Gestalt Manager to ensure that the system on which your application is running supports the Standard Mail Package.

To determine the version of the Standard Mail Package mailer functions, call the `Gestalt` function with the selector `gestaltSMPMailerVersion`. The function returns the version number of the mailers in the low-order word of the `response` parameter. For example, a value of 0x0101 indicates version 1.0.1. If the Standard Mail Package is not present and available, the `Gestalt` function returns 0 for the version number. Similarly, to determine the version of the send-letter functions, use the selector `gestaltSMPSendLetterVersion`.

Listing 3-1 shows a function that returns `true` only if the Standard Mail Package is installed and available.

Listing 3-1 Testing for the presence of Standard Mail Package services

```
Boolean MyTestForStandardMail(void)
{
   OSErr    err;
   long     response;
```

```
   err = Gestalt(gestaltSMPMailerVersion, &response);
   if ((err!=noErr) || (response==0))
      return false;

   return true;
}
```

If the Standard Mail Package is not available, you should disable those features of your application while allowing the user to use its other features normally.

After determining that the Standard Mail Package is available, you must initialize it using the `SMPInitMailer` function, passing in the version number of the package current for the services incorporated into your application. If, at run time, the current version of the Standard Mail Package is later than the one with which you compiled your application, the package initializes in compatibility mode, supporting the older version's functions. If, conversely, the run-time version is earlier, `SMPInitMailer` returns an error. The code in Listing 3-2 calls the initialization function.

Listing 3-2 Initializing the Standard Mail Package

```
OSErr MyInitStandardMail(void)
{
   OSErr err;

   SetCursor(&gWatchCursor);
   err = SMPInitMailer(kSMPVersion);
   SetCursor(&qd.arrow);
   return err;
}
```

Creating a Mailer

The Standard Mail Package enables any application to add support for mailing documents directly to other users on the network without going through intermediate mail applications. It provides standard user interface elements needed to address, send, and receive documents through the mailer, which appears as a special pane in the window of the document to be sent. This section describes how to add a mailer to a window.

Listing 3-3 uses the `SMPGetDimensions` (page 3-48) function to find the dimensions of the standard mailer window, and it creates a document window just large enough to accommodate the mailer. More typically, you would size your application windows according to the requirements of your application and use `SMPGetDimensions` to place the mailer and perform actions such as adjusting the content area of your window. The function then creates the mailer by calling the `SMPNewMailer` function (page 3-46), and it makes the mailer the initial target of user actions with the `SMPBecomeTarget`

function (page 3-54), specifying the target field within the mailer as `kSMPOther` (that is, a field other than the Recipients, From, or other enumerated field as described on page 3-32). The application-defined `MyErrorAlert` function in the listings throughout this section reports errors to the user in a standard manner.

Listing 3-3 Creating a mailer

```
void
MyBuildMailerWindow(void)
{
   Rect        boundsRect;
   Point       mailerCorner;
   short       mailerWidth;
   short       mailerContractedHeight;
   short       mailerExpandedHeight;

   boundsRect = qd.screenBits.bounds;
   boundsRect.top += ((GetMBarHeight() + 1) * 2);
   InsetRect(&boundsRect, 4, 4);
   gStatus = SMPGetDimensions(
      &mailerWidth,
      &mailerContractedHeight,
      &mailerExpandedHeight
      );
   if (gStatus != noErr)
      MyErrorAlert(gStatus, "\pSMPGetDimensions");
   else {
      boundsRect.right = boundsRect.left + mailerWidth;
      boundsRect.bottom = boundsRect.top + mailerExpandedHeight;
   }
   gMailerWindow = NewWindow(
      NULL,             /* no window storage */
      &boundsRect,      /* window shape */
      "\pMiniMailer",   /* window title */
      TRUE,             /* visible */
      documentProc,     /* document, no zoom box */
      (WindowPtr) -1L,  /* in front */
      TRUE,             /* has close box */
      0                 /* refCon (ignored) */
   );
   if (gMailerWindow == NULL) {
      MyErrorAlert(MemError(), "\pNewWindow (fatal)");
      ExitToShell();
```

```
    }
    SetPort(gMailerWindow);    /* set port to be safe */
    SetPt(&mailerCorner, 0, 0);/* locate mailer in window */
    gStatus = SMPNewMailer(    /* create Standard Mailer */
       gMailerWindow,          /* in this window */
       mailerCorner,           /* mailer top-left */
       FALSE,                  /* cannot contract */
       TRUE,                   /* initially expanded */
       0,                      /* default identity */
       nil,                    /* no prepare-to-draw callback */
       0                       /* no client data */
       );
    if (gStatus != noErr) {
       MyErrorAlert(gStatus, "\pSMPNewMailer (fatal)");
       ExitToShell();
    }
}
```

The `SMPNewMailer` function call shown in Listing 3-3 passes a value of 0 for the identity of the caller, which invokes the most recently authenticated user identity (see "Authenticating a User" on page 3-36). Note that setting the `Boolean` parameter `canContract` of the `SMPNewMailer` function to `FALSE` is unusual; Listing 3-3 does it because the window exists only to accommodate the mailer. To add a mailer to an existing document window, call the `SMPNewMailer` function, passing in the window pointer, followed by the `SMPGetDimensions` function, to adjust the size of the window content area.

Sending Mail

The first step in sending a letter is to display the send-options dialog box. This dialog box is similar to the standard print dialog box, offering the user options as to how the letter should be sent. Listing 3-4 illustrates a way to display the send-options dialog box. The code assumes that your application stores as a resource a list of formats in which it can send letters; these formats should be those in which your application can save documents. It also assumes that your application stores user preference values, including send options, in a global struct named `gPreferences`.

Listing 3-4 Displaying the send-options dialog box

```
GetResString(nativeFormat, kAppNameID, kAppName);
GetWTitle(window, docTitle);
nativeFormatArray[0] = (StringPtr)nativeFormat;
SetCursor(&qd.arrow);
err = SMPSendOptionsDialog(window, docTitle, nativeFormatArray, 1,
```

```
   kSMPNativeMask | kSMPImageMask | kSMPStandardInterchangeMask,
   &gPreferences.sendFormat, nil, 0L, &gPreferences.sendFormat,
   &gPreferences.sendOptions);
if (err == userCanceledErr)
   return;
if (err !== noErr) {
   MyErrorAlert(err, "\pSMPSendOptionsDialog");
   return;
}
```

The `SMPSendOptionsDialog` function (page 3-73) prompts the user for send options. It returns the name of the format that should be used to send the letter, which is used in the next part of the process. The process of sending a letter is begun by calling the `SMPBeginSend` function (page 3-81), passing in the user's send options (see Listing 3-5). The Standard Mail Package uses this information to build the header for the letter. Any subsequent content-adding function calls apply to the letter specified in the `SMPBeginSend` call. Listing 3-5 shows how to perform the send operation.

Listing 3-5 Performing the send operation

```
SetCursor(&gWatchCursor);
/* Use creator if you have native format, else use AppleMail. */
if ((gPreferences.sendFormat.whichFormats & kSMPNativeMask != 0) {
   letterCreator = kMyAppCreator;
   letterType = kMyLtrMsgType;
}
else {
   letterCreator = 'lap2';
   letterType = kMailLtrMsgType;
}
err = SMPBeginSend(window, letterCreator, letterType,
         &gPreferences.sendOptions, &mustAddContent);
if (err != noErr) {
   SetCursor(&qd.arrow);
   MyErrorAlert(err, "\pSMPBeginSend");
   return;
}
if (mustAddContent) {
   err = MyAddLetterBlocks(window, infoPtr,
                        &gPreferences.sendFormat);
   if (err != noErr)
      MyErrorAlert(err);
}
```

```
err = SMPEndSend(window, (err == noErr));
if (err != noErr)
   MyErrorAlert(err, "\pSMPEndSend");
```

The application-defined `MyAddLetterBlocks` function adds the actual blocks of content to the letter. It adds blocks only if the `mustAddContent` Boolean value, returned from `SMPBeginSend`, is set to `true`; there is no need to add content blocks to a letter forwarded unchanged. The function adds blocks in any combination of the three types of content formats: native application format, standard interchange (AppleMail) format, and image format. The `MyAddLetterBlocks` function calls appropriate subroutines to add the blocks.

Finally, you must call the `SMPEndSend` function (page 3-84) to send the letter. Its second parameter is a Boolean value that specifies whether to execute the send operation or to cancel the send process begun with `SMPBeginSend`. The example in Listing 3-5 uses this parameter to ensure that if `MyAddLetterBlocks` or any of its subroutines returns a nonzero error code, the send operation is canceled.

The `MyAddLetterBlocks` function and its subroutine functions are illustrated in Listing 3-6. The `MyAddLetterBlocks` function checks the `sendFormat` parameter returned from the `SMPSendOptionsDialog` function to determine which formats to add, and it calls one, two, or all three of the functions that actually add the content blocks.

Listing 3-6 Adding the letter content

```
OSErr MyAddLetterBlocks(WindowPtr window, WInfoPtr infoPtr,
         SMPSendFormat *sendFormat, StringPtr nativeFormatName)
{
   OSErr err = noErr;
   /* Add image (snapshot). */
   if (!sendFormat ||
         (sendFormat->whichFormats & kSMPImageMask)) {
      err = MyAddLetterImage(window, infoPtr);
      if (err != noErr)
         return err;
   }

   /* Add standard letter interchange format (AppleMail). */

   if (!sendFormat ||
      (sendFormat->whichFormats & kSMPStandardInterchangeMask)) {
      err = MyAddAppleMailContent(window, infoPtr);
      if (err != noErr)
         return err;
   }
```

```
   /* Add main content enclosure (native). */

   if (!sendFormat ||
         (sendFormat->whichFormats & kSMPNativeMask)) {
      err = MyAddNativeContent(window,infoPtr,nativeFormatName);
      if (err != noErr)
         return err;
   }
   return err;
}
```

Native application content is stored in files accessed by file system `FSSpec` data structures. Thus, to add native content you must save the content to a temporary file before adding it to the letter. You can use an application-defined utility routine (the `MySaveFileToTemp` function, not shown here) for this purpose. Once the temporary file is available, the `MyAddNativeContent` function (Listing 3-7) calls `SMPAddMainEnclosure` (page 3-90), passing in the letter window pointer and the file specification. Finally, the `MyAddNativeContent` function calls the `SMPAddBlock` function (page 3-91) to add a block indicating the name of the native format used in the letter.

Listing 3-7 Adding the application's native-format content

```
OSErr MyAddNativeContent(WindowPtr window, WInfoPtr infoPtr,
                         StringPtr nativeFormatName)
{
   OSErr err;
   FSSpec fSpec;
   OCECreatorType blockType;

   /* Save file temporarily so you can add by FSSpec. */

   err = MySaveFileToTemp(infoPtr, &fSpec);
   if (err != noErr)
      return err;
   err = SMPAddMainEnclosure(window, &fSpec);
   FSpDelete(&fSpec);

   /* Add native-format name string block. */

   if (err == noErr) {
      blockType.msgCreator = kMailAppleMailCreator;
      blockType.msgType = kSMPNativeFormatName;
```

```
      err = SMPAddBlock(window, &blockType, false,
                  &nativeFormatName[1], nativeFormatName[0],
                  kMailFromStart, 0);
   }
   return err;
}
```

In Listing 3-8, the `MyAddAppleMailContent` function creates and adds a content block segment in one of the AppleMail standard interchange formats. This example represents data in PICT format, indicated by the constant `kMailPictSegmentType`, passed as a parameter to the `SMPAddContent` function (page 3-85). Other standard interchange formats handle text, styled text, sound, and movies.

Listing 3-8 Adding AppleMail standard interchange-format content

```
OSErr MyAddAppleMailContent(WindowPtr window, WInfoPtr infoPtr)
{
   OSErr err;
   PicHandle thePicture;

   thePicture = MyDrawImageToPicture(window, infoPtr);
   if (thePicture) {
      HLock((Handle)thePicture);
      err = SMPAddContent(window, kMailPictSegmentType, false,
            *thePicture, GetHandleSize((Handle)thePicture),
            nil, true, smRoman);
      KillPicture(thePicture);
   }
   else return kInternalError;
   return err;
}
```

The code shown in Listing 3-9 creates an image from your document and adds it to the letter. The `SMPImage` function (page 3-88) requires you to pass in a pointer to a callback routine, an application-defined function (described on page 3-123) that actually draws the image of your document.

The `SMPImage` function adds the image blocks to the letter. You provide it with input parameters of the pointer to the letter window, a pointer to your image-drawing callback function (`MyDrawImageProc`, in Listing 3-9), a reference constant (used to pass a pointer to a block of information about the window in this example), and a Boolean value indicating whether your image-drawing function can draw in color (in Listing 3-9, it does not). The `MyDrawImageProc` function first sets up the resolution and size of the page using information in the print record for the window (in this example, a pointer to the print record is contained in the window information block passed in the reference

constant). Next, `MyDrawImageProc` calls the `SMPNewPage` function (page 3-41) to set up the graphics drawing port, as your image-drawing routine must do before drawing each page, then calls `MyDrawAllShapes` to image the page.

The application-defined `MyDrawAllShapes` function (called in Listing 3-9 but not shown) images the entire page with QuickDraw calls. The same function is called in the application-defined `MyDrawImageToPicture` function, which is used to add standard interchange format AppleMail content to a letter (see Listing 3-8). In that case the `MyDrawImageToPicture` function must provide a graphics port for QuickDraw to draw into.

Listing 3-9 Adding image-format content

```
OSErr MyAddLetterImage(WindowPtr window, WInfoPtr infoPtr)
{
   return SMPImage(window, MyDrawImageProc, (long)infoPtr, false);
}

pascal void MyDrawImageProc(long refCon, Boolean inColor)
{
   #pragma unused (inColor)
   OpenCPicParams newHeader;
   OSErr    err;
   Point    zeroPt = (0, 0);
   WInfoPtr infoPtr;
   TPrPtr   prInfo;

   infoPtr = (WInfoPtr)refCon;
   prInfo = (**(infoPtr->printRecord)).prInfo;

   newHeader.srcRect = prInfo.rPage;
   newHeader.hRes = FixRatio(prInfo.iHRes, 1);
   newHeader.vRes = FixRatio(prInfo.iVRes, 1);
   newHeader.version = -2;
   newHeader.reserved1 = 0;
   newHeader.reserved2 = 0L;
   err = SMPNewPage(&newHeader);
   if (err != noErr)
      MyErrorAlert(err, "\pSMPNewPage");
   MyDrawAllShapes(infoPtr, zeroPt);
}
```

Receiving Mail

A mail-aware application can receive mail in either of two ways: the user can double-click the letter in the mailbox In Tray or in the Finder. In either case, this action generates an Open Documents core Apple event (`'aevt' 'odoc'`) that the Finder sends to your application. If the letter is on disk, the Apple event includes a file specification of type `FSSpec`; if it is in the In Tray, the Apple event includes instead a letter specification of type `LetterSpec`. The portion of an Apple event handler shown in Listing 3-10 shows how to process both file and letter specifications. The Standard Mail Package handles both file and letter specifications through the letter descriptor structure, which includes both formats.

Listing 3-10 Apple event handler processing both file and letter specifications

```
AECountItems(&docList, &itemsInList);
for (index = 1; index <= itemsInList; index++) {
   err = AESizeOfNthItem(&docList, index, &returnedType, &size);
   if (err != noErr)
      return err;
   if (returnedType == typeLetterSpec) {
      diskForm = false;
      err = AEGetNthPtr(&docList, index, typeLetterSpec, &keywd,
            &returnedType, (Ptr)&myLetterSpec, sizeof(LetterSpec),
            &actualSize);
   } else if ((returnedType == typeAlias) ||
                              (returnedType == typeFSS)) {
      diskForm = true;
      err = AEGetNthPtr(&docList, index, typeFSS, &keywd,
            &returnedType, (Ptr)&myFSS, sizeof(myFSS),
            &actualSize);
   }
   if (err != noErr)
         return err;
   if ((returnedType == typeLetterSpec) ||
         (returnedType == typeAlias) ||
         (returnedType == typeFSS)) {
      err = MyHandleOpenDoc(diskForm, &myFSS, &myLetterSpec);
      if (err != noErr)
         return err;
   }
}
```

The `MyHandleOpenDoc` function shown in Listing 3-11 uses this information to open a letter in the mailbox or on disk. The `SMPOpenLetter` function (page 3-94) registers with

the Standard Mail Package the window passed to it and associates it with the letter identified in the `LetterDescriptor` structure. The `SMPGetMainEnclosureFSSpec` function (page 3-103) then extracts the native format document from the letter, and an application-defined content-drawing routine (`MyDrawLetterContent`, in this example) draws the document into the window.

Listing 3-11 Opening a letter

```
OSErr MyHandleOpenDoc(Boolean diskForm, FSSpec *myFSS,
                                  LetterSpec *myLetterSpec)
{
   OSErr             err;
   LetterDescriptor  letterDesc;
   Point             upLeft = (0, 0);
   Rect              newWindowRect;

   letterDesc.diskForm = diskForm;
   if (diskForm)
   {
      letterDesc.fileSpec = *myFSS;
   }
   else
   {
      letterDesc.fileSpec = *myLetterSpec;
   }

   newWindow = MyMakeWindow(kDrawMailerWindow, &newWindowRect,
                                                "\pTitle", false);
   if (newWindow == NULL)
   {
      MyErrorAlert(memFullErr, "\pSMPOpenLetter");
      return memFullErr;
   }

   err = SMPOpenLetter(&letterDesc, newWindow, upLeft, true,
                           gPreferences.expandOnOpen, nil, 0L);
   if (err != noErr)
   {
      MyErrorAlert(err, "\pSMPOpenLetter");
      return err;
   }

   err = SMPGetMainEnclosureFSSpec(newWindow, &enclSpec);
```

```
   if (err != noErr)
   {
      MyErrorAlert(err, "\pSMPOpenLetter");
      return err;
   }

   return MyDrawLetterContent(newWindow, &enclSpec);
}
```

Forwarding and Replying to Mail

After opening a letter, the user has the option to reply or to forward it. The user can also remove the mailer, changing the letter into a regular document.

To forward a letter, you must add a new mailer to the existing letter. A letter has a new mailer attached each time it is forwarded. The mailers form a set with each mailer superimposed upon the preceding mailers. The user can view the mailers in the set by clicking a dog-ear in the corner of the mailer window pane to cycle through the set or by choosing among the names in a pop-up menu appearing in the Forwarded By field.

The first step in forwarding a letter is to expand the existing mailer, if it is contracted. Next, you call the `SMPMailerForward` function (page 3-49) to create the new mailer and add it to the letter. Finally, you should adjust your menu items in the configuration appropriate for sending mail, which is done in Listing 3-12 by an application-defined function `MyFixMailerMenus`. The parameter constant `kDefaultIdentity` has a value of 0, with the effect described in "Authenticating a User" on page 3-36.

Listing 3-12 Forwarding a letter

```
err = SMPExpandOrContract(window, true);
/* Ignore errors if window is already expanded. */
err = SMPMailerForward(window, kDefaultIdentity);
if (err != noErr)
   MyErrorAlert(err, "\pSMPMailerForward");
MyFixMailerMenus(window);
```

The first step in replying to a letter is to create a new window in which the user will write the reply. When this new window exists, you can call the `SMPMailerReply` function (page 3-51), passing in among other parameters the new window and the existing letter window. The function causes the reply letter to be created, automatically addressed to the originator of the original letter. The code shown in Listing 3-13 illustrates how to handle replying to a letter.

Listing 3-13 Replying to a letter

```
replyWindow = MyMakeWindow(kDrawMailerWindow, &newWindowRect,
                  newTitle, false);
err = SMPMailerReply(window, replyWindow, replyToAll, topLeft,
                  true, true, kDefaultIdentity, nil, 0L);
if (err != noErr)
   MyErrorAlert(err, "\pSMPMailerReply");
ShowWindow(replyWindow);
```

The application-defined function `MyMakeWindow` creates a window and adjusts its content area to accommodate the mailer. The `SMPMailerReply` function adds the mailer, and the `ShowWindow` function causes the window to become visible.

Closing a Letter

Closing a letter window requires you to adhere to a short procedure: displaying the close-options dialog box, checking for open enclosures and in-progress copy operations, removing the mailer from the window, and closing the window.

Before closing a letter window, you can display the close-options dialog box, which gives the user an opportunity to delete the letter or tag it before closing it. Listing 3-14 assumes the existence of a data structure `gPreferences` containing user-preference flags, including one determining whether or not you should display the close-options dialog box. The code uses these preferences also to fill in the default values in the close-options dialog box when it is displayed by the `SMPCloseOptionsDialog` function (page 3-60).

Listing 3-14 Preparing to close a letter

```
if (gPreferences.closeOptionsDialog) {
   SetCursor(&qd.arrow);
   err = SMPCloseOptionsDialog(window
                     &gPreferences.closeOptions);
   if (err != noErr)
      returnValue = false;
}
```

The next step in the letter-closing procedure is to ensure that there are no open enclosures attached to the letter, that there are no Finder copy operations in progress, and that there are no other conditions that prevent closing the window. Finder copy operations occur when the user is in the process of copying a document to or from the enclosures list. If either situation is true, or if for some other reason a nonzero result was returned from the `SMPPrepareToClose` function, the application-defined function `MyStopAlert` notifies the user and prevents the letter from closing. Listing 3-15 illustrates these checks.

Listing 3-15 Checking status prior to closing a letter

```
err = SMPPrepareToClose(window);
if (err == kSMPHasOpenAttachments) {
   SetCursor(&qd.arrow);
   MyStopAlert(kMyHasOpenAttachID, nil);
   returnValue = false;
}
else if (err == kSMPCopyInProgress) {
   SetCursor(&qd.arrow);
   MyStopAlert(kMyCopyInProgress, nil);
   returnValue = false;
}
else if (err != noErr) {
   SetCursor(&qd.arrow);
   MyStopAlert(kMyCannotCloseWindow, nil);
   returnValue = false;
}
```

The final steps in the closing procedure are to remove the mailer from the window and close the window, as shown in Listing 3-16. The `SMPDisposeMailer` function (page 3-61) removes the mailer from the window passed in as a parameter and releases the memory associated with the letter window. Then the application-defined routine `MyDestroyWindow` disposes of the rest of the window and document structures in memory.

Listing 3-16 Closing the letter

```
err = SMPDisposeMailer(window, closeOptions);
if (err != noErr)
   MyErrorAlert(err, "\pSMPDisposeMailer");
return MyDestroyWindow(window);
```

Handling Mailer Events

The general strategy for handling events in a window with a mailer is to hand the events to the Standard Mail Package first. The Standard Mail Package has built-in routines to handle many events, including mouse-down events, key-down events, update events for the mailer, activate events, deactivate events, and null events. The Standard Mail Package then hands the event back to the application with an indication that either it handled the event or your application must handle the event.

Your application should retrieve events in the normal manner, with the `WaitNextEvent` system call. When a mailer window is frontmost, call the `SMPMailerEvent` function (page 3-63), passing in the event record. The

SMPMailerEvent function returns a set of flags in its whatHappened parameter indicating what action it took, if any, and whether your application must handle the event. (These flags, of type SMPMailerResult, are described on page 3-65.) Your application can then process the event appropriately. The event-handling function shown in Listing 3-17 receives the event record following a WaitNextEvent call, calls SMPMailerEvent, and passes the SMPMailerResult value to an application-defined routine named MyProcessWhatHappened. The parameter of type WInfoPtr is a pointer to an application-defined data structure containing status information about the mailer window.

Listing 3-17 Processing events in a mailer window

```
void *MyMailerEventHandler(WindowPtr window, WInfoPtr infoPtr,
                           EventRecord *ev)
{
   SMPMailerResult   whatHappened;
   OSErr             err;

   err = SMPMailerEvent(ev, &whatHappened, nil, 0L);
   if (err != noErr)
      MyErrorAlert(err, "\pSMPMailerEvent");
   return (void *)(MyProcessWhatHappened(window, infoPtr,
                                         whatHappened));
}

Boolean MyProcessWhatHappened(WindowPtr window, WInfoPtr infoPtr,
                              SMPMailerResult whatHappened)
{
   OSErr          err;
   SMPMailerState state;
   long           *lastChanged;

   /* Check if mailer has changed since last menu adjustment. */
   err = SMPGetMailerState(window, &state);
   if (err != noErr)
      MyErrorAlert(err, "\pSMPGetMailerState");
   lastChanged = (long *)&infoPtr->otherData[kLastChangedData];
   if (*lastChanged != state.changeCount) {
      *lastChanged = state.changeCount;
      infoPtr->changed = true;
      MyFixMailerMenus(window, infoPtr);
   }
   if ((whatHappened & kSMPContractedMask) != 0)
```

```
      MyHandleContract(window,infoPtr);
   if ((whatHappened & kSMPExpandedMask) != 0)
      MyHandleExpand(window, infoPtr);
   if ((whatHappened & kSMPMailerBecomesTargetMask) != 0 ||
         (whatHappened & kSMPAppBecomesTargetMask) != 0))
      MyFixMailerMenus(window, infoPtr);
   /* Check menus for every event the mailer handles. */
   if ((whatHappened & kSMPAppShouldIgnoreEventMask) != 0)
      MyFixMailerMenus(window, infoPtr);
   if ((whatHappened & kSMPAppMustHandleEventMask) != 0)
      return false; /* app must handle this event */
   else return true; /* mailer handled this event completely */
}
```

Most of the postprocessing of the event involves adjusting the menus, because the mailer event may have affected which commands should be active. In addition, if the `kSMPContractedBit` flag or `kSMPExpandedBit` flag is set as a a result of the event, the code calls one of the application-defined routines: `MyHandleContract` or `MyHandleExpand`. These routines call the `SMPGetDimensions` function to determine the size of the expanded or contracted mailer, so that the application can adjust the size of the content region of the window. If the user wants to expand the mailer, you must then call the `SMPExpandOrContract` function (page 3-56) to expand the mailer to its full size. However, if the user wants to contract the mailer to a single line, you need not call `SMPExpandOrContract` because the Standard Mail Package performs the contraction; you need only adjust the size of your content region and invalidate it to update its content.

In addition, the Standard Mail Package requires you to add some logic to your application's mouse-click handler for a window that includes a mailer. You must notify the Standard Mail Package before you allow the user to change the content of a letter, to accommodate the needs of its digital signature capability. Before changing the letter, you must call the `SMPPrepareToChange` function (page 3-83); if the letter has been digitally signed, a dialog box appears warning the user that the impending change will invalidate the signature. As in Listing 3-18, your routine should check the return value from `SMPPrepareToChange` and exit if the user has clicked the Cancel button in the dialog box.

The Standard Mail Package maintains its own undo buffer to support undoing mailer operations. You must clear this buffer before doing operations on data in the content area of your window so that only one undo operation is pending for the window. After calling the application's click-handler function, if the letter's contents have changed, you should call the `SMPContentChanged` function (page 3-76).

Listing 3-18 Handling a mouse click in a mailer window

```
void *MyMailerMouseClickHandler(WindowPtr window,
                                WInfoPtr infoPtr)
{
   void        *returnVal;
   OSErr       err;
   Boolean     alreadyChanged;

   /* Make sure you can change the letter. */
   alreadyChanged = infoPtr->changed;
   if (!alreadyChanged) {
      err = SMPPrepareToChange(window);
      if (err == userCanceledErr)
         return nil;
   }

   /* Since content is changing, clear mailer undo buffer. */
   err = SMPClearUndo(window);
   if (err != noErr)
      MyErrorAlert(err, "\pSMPClearUndo");

   /* Call app's click handler. */
   returnVal = MyClickHandler(window, infoPtr);
   if (!alreadyChanged && infoPtr->changed) {
      err = SMPContentChanged(window);
      if (err != noErr)
         MyErrorAlert(err, "\pSMPContentChanged");
   }

   return returnVal;
}
```

The previous section alluded to the undo buffer kept by the Standard Mail Package to support undo operations in the mailer portion of letters. The Standard Mail Package supports the Clipboard-based edit commands Cut, Copy, Paste, Clear, Select All, as well as the Undo command. The function shown in Listing 3-19 is a mailer Cut command handler; it shows how to support the Clipboard by calling the `SMPMailerEditCommand` function (page 3-67), then processing the result by calling the application-defined `MyProcessWhatHappened` function. You can use a similar strategy for the Copy, Paste, Clear, Select All, and Undo commands.

Listing 3-19 Supporting the Clipboard in a mailer edit command

```
void *MyMailerCutCommand(WindowPtr window, WInfoPtr infoPtr)
{
   OSErr             err;
   SMPMailerResult   whatHappened;

   err = SMPMailerEditCommand(window, kSMPCutCommand,
                                          &whatHappened);
   if (err != noErr)
      MyErrorAlert(err, "\pSMPMailerEditCommand");
   return (void *)(MyProcessWhatHappened(window, infoPtr,
                                              whatHappened));
}
```


Standard Mail Package Reference

This section describes the data types and routines provided by the Standard Mail Package.

Data Structures

The Standard Mail Package routines use the data types described in this section.

Recipient Descriptor

The recipient descriptor, used by the `SMPSendLetter` and `SMPResolveToRecipient` functions, describes an addressee for a message or letter.

Note
You must call the `DisposePtr` function to deallocate the `recipient` field before you can dispose of the recipient descriptor. ◆

```
struct SMPRecipientDescriptor
{
   struct SMPRecipientDescriptor *next;      /* pointer to next element */
   OSErr                         result;     /* result code */
   OCEPackedRecipient            *recipient; /* packed recipient address */
   unsigned long                 size;       /* size of recipient address */
   MailRecipient                 theAddress; /* unpacked recipient address */
   RecordID                      theRID;     /* record ID of recipient */
};
```

Field descriptions

`next`	A pointer to the next element in a linked list of recipient descriptors. This field must be set to `nil` in the last descriptor in the list.
`result`	The result code returned by the `SMPSendLetter` function. If the `SMPSendLetter` function fails because of a bad recipient descriptor, you can examine this field in each of the recipient descriptors to determine which caused the problem.
`recipient`	A pointer to the packed address of the recipient of the letter.
`size`	The length, in bytes, of the recipient's address.
`theAddress`	The unpacked address of the recipient.
`theRID`	The record ID of the recipient. If the `SMPSendLetter` function fails because of a bad recipient descriptor, you can use this record ID to determine the name of the addressee that caused the error.

Enclosure Descriptor

The enclosure descriptor is an element of a linked list that describes an enclosure to be sent with a letter. See the description of the `SMPSendLetter` function on page 3-37 for more information about the use of this data structure.

```
struct SMPEnclosureDescriptor
{
   struct SMPEnclosureDescriptor *next;         /* pointer to next element */
   OSErr                         result;        /* result code */
   FSSpec                        fileSpec;      /* file specifier  */
                                                /*  of enclosure */
   OSType                        fileCreator;   /* creator of enclosure */
   OSType                        fileType;      /* file type of enclosure */
};
```

Field descriptions

`next`	A pointer to the next element in the linked list. If this is the only or last element in the list, set this field to `nil`. If you use the `SMPResolveToRecipient` function to create the linked list, the function fills in this field for you.
`result`	The result code returned by the `SMPSendLetter` function. If the `SMPSendLetter` function fails because of a bad enclosure descriptor, you can examine this field in each of the enclosure descriptors to determine which caused the problem.
`fileSpec`	File specifier of the enclosure.
`fileCreator`	File creator of the enclosure. The `SMPSendLetter` function uses this field only if you send the enclosure directly as a file (that is, you set the `sendAs` field of the parameter block for the `SMPSendLetter` function to `kSMPSendFileOnlyMask`).

`fileType` File type of the enclosure. The `SMPSendLetter` function uses this field only if you send the enclosure directly as a file (that is, you set the `sendAs` field of the parameter block for the `SMPSendLetter` function to `kSMPSendFileOnlyMask`).

Letter Descriptor

The letter descriptor, used by the `SMPOpenLetter` (page 3-94) and `SMPGetNextLetter` (page 3-97) functions, identifies a letter in the In Tray or on disk.

```
struct LetterDescriptor {
   Boolean onDisk;
   union {
      FSSpec fileSpec;
      LetterSpec mailboxSpec;
   }u;
};
```

Field descriptions

`onDisk` A Boolean value that indicates whether the letter is on disk or in the In Tray. If this value is set to `true`, the file is on disk.

`fileSpec` The file specification structure of the letter. Use this field of the structure if the file is on disk.

`mailboxSpec` The letter specification structure of the letter. Use this field if the letter is in the In Tray. When the user double-clicks a letter in the In Tray and the letter's creator is your application, you receive an `'aevt' 'odoc'` Apple event that includes this specifier. The `LetterSpec` structure is defined on page 3-35.

Letter Information Structure

The letter information structure, which is used by the `SMPGetLetterInfo` (page 3-93) function, describes a letter in the In Tray.

```
struct SMPLetterInfo {
   OSType      letterCreator;
   OSType      letterType;
   RString32   subject;
   RString32   sender;
};
```

Field descriptions

`letterCreator` The creator of the letter. The field indicates what application created the letter and is identical to the creator used by the application for files.

`letterType`	The letter type, which is identical to the file type that the creating application would use for the letter. Letters containing only AOCE standard content are of type `'lttr'`.
`subject`	The contents of the Subject field in the mailer.
`sender`	The contents of the From field in the mailer.

Creator Type Structure

The Standard Mail Package uses the creator type structure to specify block types. The creator type structure is defined by the `OCECreatorType` data type.

```
struct OCECreatorType {
   OSType   msgCreator;    /* block creator */
   OSType   msgType;       /* block type */
};
```

Field descriptions

`msgCreator`	The creator of the block. You can specify any four-character value in this field; usually it is the signature of your application that adds the block of data to the letter. For example, the creator of a block added by the AppleMail application provided with the AOCE software is `'apml'`.
`msgType`	The type of the block. You can define your own four-character block types to serve your own purposes. Apple Computer, Inc., reserves all block types consisting entirely of lowercase letters. For example, the type of an image block as defined by the AppleMail application is `'imag'`.

Image Block Information Structure

An image block in a letter (a block with a creator type of `'apml'` and a block type of `'imag'`) starts with an image block information structure, defined by the `TPfPgDir` data type (defined by the Printing Manager).

```
struct TPfPgDir{
   short    iPages;        /* number of pages in image block */
   long     iPgPos[129];   /* array [0..iPfMaxPgs] of offsets */
};
```

Field descriptions

`iPages`	The number of pages in the image. The image block contains one PICT resource for each page.
`iPgPos`	An array of offsets from the start of the block to the picture elements that follow the `TPfPgDir` structure.

The `iPgPos` array contains offsets to the picture elements that follow the `TPfPgDir` structure. The offset from the start of the image block to the image of page $n + 1$ is `iPgPos[`n`]` (because page numbers start at 1 and the array elements start at 0). The array contains `iPgPos[`$n + 1$`]` elements for a document of n pages. The last element is the offset of the end of the last page from the beginning of the block. You can determine the size of a page by subtracting the offset of the current page from the offset of the next page, that is, the size of page n is `iPgPos[`n`]` – `iPgPos[`$n - 1$`]`.

Allocate and lock down a buffer equal to the size of the page. Then call the `SMPReadBlock` function (page 3-106) with the pointer to that buffer in the `buffer` parameter and the offset `iPgPos[`$n - 1$`]` in the `dataOffset` parameter.

Letter Parameter Block

The `SMPSendLetter` function uses the `SMPLetterPB` parameter block. The fields of the parameter block are described with the `SMPSendLetter` function on page 3-38.

```
struct SMPLetterPB
{
   OSErr                       result;        /* function result */
   RStringPtr                  subject;       /* subject of letter */
   AuthIdentity                senderIdentity;/* identity of sender */
   SMPRecipientDescriptorPtr   toList;        /* list of addressees */
   SMPRecipientDescriptorPtr   ccList;        /* list of cc addressees */
   SMPRecipientDescriptorPtr   bccList;       /* list of bcc addressees */
   ScriptCode                  script;        /* script code for language */
   Size                        textSize;      /* length of body data */
   Ptr                         textBuffer;    /* body of the letter */
   SMPPSendAs                  sendAs;        /* file, enclosure, or image */
   Byte                        padByte;       /* reserved */
   SMPEnclosureDescriptorPtr   enclosures;    /* files to be enclosed */
   SMPDrawImageProcPtr         drawImageProc; /* your imaging routine */
   long                        imageRefCon;   /* for your use */
   Boolean                     supportsColor; /* true for a color grafPort */
};
```

Close-Options Structure

The `SMPCloseOptionsDialog` function (page 3-60) and the `SMPDisposeMailer` function (page 3-61) use the close-options structure to specify what actions the Standard Mail Package should take when the user closes a letter in the In Tray. The close-options structure is defined by the `SMPCloseOptions` data type.

```
struct SMPCloseOptions {
   Boolean              moveToTrash;
   Boolean              addTag;
   RString32            tag;
};
```

Field descriptions

`moveToTrash`	Move the letter from the In Tray to the Trash. You should not set this field to `true` if the `addTag` field is set to `true`.
`addTag`	Tag the letter with the value in the `tag` field. You should not set this field to `true` if the `moveToTrash` field is set to `true`.
`tag`	The tag to attach to the letter. This field must contain a valid tag if the `addTag` field is set to `true`. A tag can be any alphanumeric string up to 32 bytes in length.

Mailer-State Structure

The `SMPGetMailerState` function (page 3-69) uses the mailer-state structure to return information about a mailer in a specified window. The mailer-state structure is defined by the `SMPMailerState` data type.

```
struct SMPMailerState {
   short                mailerCount;
   short                currentMailer;
   Point                upperLeft;
   Boolean              hasBeenReceived;
   Boolean              isTarget;
   Boolean              isExpanded;
   Boolean              canMoveToTrash;
   Boolean              canTag;
   Byte                 padByte2;
   unsigned long        changeCount;
   SMPMailerComponent   targetComponent;
   Boolean              canCut;
   Boolean              canCopy;
   Boolean              canPaste;
   Boolean              canClear;
   Boolean              canSelectAll;
   Byte                 padByte3;
   SMPUndoState         undoState;
   Str63                undoWhat;
};
```

Field descriptions

`mailerCount` The number of mailers in the mailer set associated with the window. This number is incremented by 1 each time the letter is forwarded. You should enable the Reply item in the Mail menu if the `mailerCount` field is greater than 1.

`currentMailer` The number of the mailer that the user is currently looking at. The original mailer for the letter is number 1, and each forwarding mailer is numbered sequentially.

`upperLeft` The upper-left corner of the mailer in the window's local coordinates.

`hasBeenReceived`
A Boolean value that indicates whether the most recent mailer has been received (that is, it was sent to the current user). If set to `true` (that is, if the mailer has been received), then the user cannot edit the fields in the mailer but can forward or reply to the letter. You should enable the Forward and Reply items in the Mail menu. If it is set to `false`, the current user is the originator of the letter or has added a new mailer to forward the letter, and might still be working on the letter, so you should disable the Forward item.

`isTarget` A Boolean value that indicates whether the mailer is the target; that is, whether the user is working in the mailer so that key-down events apply to the mailer rather than to the portion of the window that you control. Note that the Event Manager sends all events that take place in your window— including in the mailer—to your application. If you pass every event to the `SMPMailerEvent` function (page 3-63), that function returns a value that tells you whether you have to handle the event.

`isExpanded` A Boolean value that indicates whether the mailer is in the expanded state or contracted state.

`canMoveToTrash`
A Boolean value that indicates whether to enable the Close and Delete item in the File menu. The standard interface is to enable this item for a letter that is in the In Tray, but not for one that has been saved to disk.

`canTag` A Boolean value that indicates whether to enable the Tag item in the Mail menu. The user can add a tag to a letter that is in the In Tray, but not to a letter that has been saved to disk. See the `SMPTagDialog` function (page 3-58) to see how to implement the Tag item in the Mail menu.

`changeCount` A value that indicates whether the mailer has been changed. If this field is set to a nonzero value, the mailer has been changed since the last time it was saved. If this number has changed since the last time you checked it, then the mailer has been changed during that period.

`targetComponent`
A constant that indicates which of the fields in the mailer the user is working in. Possible values for this field are listed immediately following these field descriptions.

`canCut`	A Boolean value that indicates whether you should enable the Cut item in the Edit menu. This field is significant only if the `isTarget` field is set to `true`.
`canCopy`	A Boolean value that indicates whether you should enable the Copy item in the Edit menu. This field is significant only if the `isTarget` field is set to `true`.
`canPaste`	A Boolean value that indicates whether you should enable the Paste item in the Edit menu. This field is significant only if the `isTarget` field is set to `true`.
`canClear`	A Boolean value that indicates whether you should enable the Clear item in the Edit menu. This field is significant only if the `isTarget` field is set to `true`.
`canSelectAll`	A Boolean value that indicates whether you should enable the Select All item in the Edit menu. This field is significant only if the `isTarget` field is set to `true`.
`undoState`	A constant that you can use to determine whether you should enable the Undo item in the Edit menu. See the description of the `SMPClearUndo` function on page 3-70 for information on clearing the undo buffer. The possible values for this field are described following these field descriptions.
`undoWhat`	A string that indicates the action that the reader should undo or redo. You should use this string in place of the word "Undo" or "Redo" in the Edit menu. For example, if the user just used the Edit menu to cut a word from the subject field in the mailer, the `undoWhat` field is set to the string `Undo Cut`. This field is significant only if the `undoState` field equals `kMailerUndo`.

Here are the possible values for the `targetComponent` field. These values are also used by the `SMPBecomeTarget` function (page 3-54), the `SMPGetComponentSize` function (page 3-110), the `SMPGetComponentInfo` function (page 3-111), and the `SMPGetListItemInfo` function (page 3-113).

```
enum {
   kSMPOther         = -1,
   kSMPFrom          = 32,
   kSMPTo            = 20,
   kSMPRegarding     = 22,
   kSMPSendDateTime  = 29,
   kSMPAttachments   = 26,
   kSMPAddressOMatic = 16
};

typedef unsigned long SMPMailerComponent;
```

Constant descriptions

`kSMPOther` No field, or some field other than those indicated by the other enumerated values (such as the Signature field).

`kSMPFrom` The From field in a mailer for a new letter, or the Forwarded By field in a mailer for a forwarded letter.

`kSMPTo` The Recipients field.

`kSMPRegarding` The Subject field.

`kSMPSendDateTime`
The Sent field.

`kSMPAttachments`
The Enclosures field.

`kSMPAddressOMatic`
The addressing panel (see Figure 3-4 on page 3-6).

Your application and the mailer maintain independent undo buffers. The mailer keeps track of which undo buffer should currently be in use and passes this information to you in the `undoState` field of the mailer-state structure. You can use this information to determine which items in the Edit menu to enable and whether to clear your application's undo buffer. The possible values for the `undoState` field are as follows:

```
enum {
   kSMPUndoDisabled,
   kSMPAppMayUndo,
   kSMPMailerUndo
};

typedef unsigned short SMPUndoState;
```

Constant descriptions

`kSMPUndoDisabled`
The Standard Mail Package has cleared its undo buffer after executing a command that the user cannot undo. Therefore, there is currently no action in the mailer or in your application that the user can undo. You should disable the Undo item in the Edit menu and clear your application's undo buffer.

`kSMPAppMayUndo`
The Standard Mail Package has not executed a command that the user may undo. Therefore, there is no action in the mailer that the user can undo, but the Standard Mail Package can't tell whether there is an action in your application that the user can undo. You should enable the Undo item in the Edit menu only if your application has executed a command that the user may undo. If the user has taken an action in the content portion of the window that the user can undo or that should cause the undo buffer to be cleared, you must also call the `SMPClearUndo` function (page 3-70) to tell the Standard Mail Package to clear its undo buffer.

`kSMPMailerUndo`

The Standard Mail Package has executed a command that the user may undo. Therefore, the latest action that the user can undo was in the mailer, and there is no action in your application that the user can undo. You should enable the Undo item in the Edit menu and display the string returned in the `undoWhat` field of the `SMPMailerState` structure. You should also clear your application's undo buffer. If the user chooses the Undo item in the Edit menu, call the `SMPMailerEditCommand` function to allow the Standard Mail Package to handle the undo operation.

Send-Options Structure

The Standard Mail Package maintains a set of options for each letter. There is a default value for each option, but before you send a letter, you should give the user the opportunity to change the send options for that letter. You can call the `SMPSendOptionsDialog` function (page 3-73) to provide the user with a dialog box that sets these options. The `SMPSendOptionsDialog` function returns the send-options structure, defined by the `SMPSendOptions` data type.

```
struct SMPSendOptions {
   Boolean     signWhenSent;
   IPMPriority priority;
};
```

Field descriptions

`signWhenSent` A Boolean value that indicates whether a digital signature should be added to the letter when you send it. If this field is set to `true`, the Standard Mail Package prompts the user for a signature when you send the letter.

`priority` A constant that indicates the priority of the message. The Standard Mail Package includes the priority information in the In and Out Trays.

Send-Format Structure

The Standard Mail Package uses two standard formats and allows applications to send or open letters in any number of "native" formats known to the application. The two standard formats used by the Standard Mail Package are standard interchange format and image format. Native formats for the SurfWriter word processing program might be SurfWriter, TIFF, and SGML, for example.

Before you send a letter using the Standard Mail Package, you call the `SMPSendOptionsDialog` function (page 3-73). This function displays a dialog box that lets the user indicate which format or formats to use when sending the letter and returns the user's choices to you in a send-format structure.

The send-format structure includes a `whichFormats` field that indicates whether you should send the document in a format designed to be read by your application (`kSMPNativeBit`), as an image designed to be read by any application that reads AOCE image files (`kSMPImageBit`), or in standard interchange format (`kSMPStandardInterchangeBit`).

```
enum {
   kSMPNativeBit,
   kSMPImageBit,
   kSMPStandardInterchangeBit
};

/* values of SMPSendFormatMask */
enum {
   kSMPNativeMask                = 1<<kSMPNativeBit,
   kSMPImageMask                 = 1<<kSMPImageBit,
   kSMPStandardInterchangeMask   = 1<<kSMPStandardInterchangeBit,
};
typedef unsigned long SMPSendFormatMask;
```

The send-format structure is defined by the `SMPSendFormat` data type.

```
struct SMPSendFormat {
   SMPSendFormatMask    whichFormats;
   short                whichNativeFormat;   /* 0 based */
};
```

The `whichNativeFormat` field is an index number (starting with 0) that indicates which one of your application's native formats has been selected by the user, or, in the case of a received letter, which native format is currently in the letter. The index number refers to the array of string pointers you pass to the `SMPSendOptionsDialog` function in the `nativeFormatNames` parameter. The `whichNativeFormat` field is significant only if the `whichFormats` field has the `kSMPNativeBit` set to 1.

Letter-Specification Structure

The letter-specification structure is a data structure that you receive from an `'aevt' 'odoc'` Apple event and pass to the `SMPOpenLetter` function (page 3-94). The content of this data structure is private to the AOCE toolbox.

```
struct LetterSpec
{
   unsigned long     spec[3];
};
```

Standard Mail Package Functions

The following sections describe the routines provided by the AOCE Standard Mail Package. Several Standard Mail Package routines require you to provide an authentication identity as input. The chapter "Standard Catalog Package" in this book describes a routine that prompts the user for a name and password, authenticates the user, and returns the authentication identity number to your application.

The routines in this chapter are divided into two main sections, reflecting the two parts of the Standard Mail Package:

- Send-letter functions, which provide a very simple way to send a letter or a file.
- Mailer functions, which provide a standard user interface for sending and opening your application's documents as letters.

A final section, "Application-Defined Functions," describes some callback routines that you can provide to support Standard Mail Package features.

Assembly-Language Interface

To call a Standard Mail Package routine from assembly language, you must do the following:

1. Push space for the function result and all routine parameters (in Pascal calling-convention order) on the stack.
2. Put in the D0 register a long word consisting of the parameter word count for the routine followed by the routine selector. The parameter word count indicates how many words of parameters you are placing on the stack; for example, if the function has two parameters and each is a pointer, the parameter word count for the function is $0004.
3. Call the Standard Mail Package trap, $AA5D.

Each routine description in the following sections lists the parameter word count and routine selector for that routine.

Authenticating a User

Before the first time you send a message, you must provide identification to prove that the caller is an authorized user of the system. The `SDPPromptForID` function described in the chapter "Standard Catalog Package" in this book provides dialog boxes that allow the user to identify himself or herself as one of the authorized users of the system and returns an identification number (the *authentication identity*) for the user. You can use the authentication identity in all subsequent calls to Standard Mail Package and other AOCE routines that require it.

However, the Standard Mail Package implements a special scheme to ease handling authentication identities for its routines. That is, you can pass a value of 0 for the authentication identity parameter to those functions requiring it. The effect of passing the 0 parameter value varies according to the situation. The first time you pass 0 after initializing the Standard Mail Package, the system uses the local identity (see the chapter "Standard Catalog Package" in this book for a description of local and specific identities).

If you forward or reply to a letter, the Standard Mail Package uses the identity for the mailbox the letter was in: a visitor's mailbox produces the visitor's identity; the main mailbox produces the local identity. In all other cases, if you pass 0 for the authentication identity parameter, the Standard Mail Package uses the last identity (local or specific) used by the user.

Send-Letter Functions

You can use the functions in this section to send a document as a letter with enclosures, as an image, or as a file. The `SMPSendLetter` function sends the document. If you want to send the document as an image, you must provide an image-drawing routine that calls the `SMPNewPage` function each time it images a page of the document.

You can obtain catalog system specification (`DSSpec`) structures for the recipients of the letter by using the dialog boxes or the Catalog-Browsing panel described in the chapter "Standard Catalog Package" in this book. You can use the `SMPResolveToRecipient` function described in this section to transform the `DSSpec` structures into a linked list of mail addresses, and you can use this linked list as input to the `SMPSendLetter` function.

The `SMPSendLetter` function includes as a parameter a pointer to a parameter block. The routine description includes a list of the parameter block fields for which you must provide values or that return values to you. Each parameter block field list in the routine description consists of four columns, as described in the Preface of this book.

SMPSendLetter

The `SMPSendLetter` function sends a letter, an image, or a file.

```
pascal OSErr SMPSendLetter(SMPLetterPBPtr theLetter);
```

`theLetter` Pointer to a parameter block.

Parameter block

←	result	OSErr	Result code
→	subject	RStringPtr	Subject of letter
→	senderIdentity	AuthIdentity	Identity of sender
→	toList	SMPRecipientDescriptorPtr	List of recipients
→	ccList	SMPRecipientDescriptorPtr	List of cc recipients
→	bccList	SMPRecipientDescriptorPtr	List of bcc recipients
→	script	ScriptCode	Script code
→	textSize	Size	Length of text
→	textBuffer	Ptr	Letter text
→	sendAs	SMPPSendAs	Letter, image, or file
→	enclosures	SMPEnclosureDescriptorPtr	Enclosed files
→	drawImageProc	SMPDrawImageProcPtr	Image-drawing routine
→	imageRefCon	long	For your use
→	supportsColor	Boolean	Set to `true` for a color graphics port

Field descriptions

`result` The function result. This field contains the same result code as the function return value.

`subject` The subject string for the letter.

`senderIdentity`
Authentication identity of the sender.

`toList` A pointer to a linked list of recipient descriptors for the main addressees of the letter. You can use the `SMPResolveToRecipient` function to create this list.

`ccList` A pointer to a linked list of recipient descriptors for the "carbon copy" (cc) addressees of the letter. You can use the `SMPResolveToRecipient` function to create this list.

`bccList` A pointer to a linked list of recipient descriptors for the "blind carbon copy" (bcc) addressees of the letter. You can use the `SMPResolveToRecipient` function to create this list.

`script` Language of letter text. This is a script code from the Script Manager. You cannot use the values `smSystemScript` or `smCurrentScript` for this parameter. The function ignores this field if you set the `sendAs` field to `kSMPSendFileOnlyMask`.

`textSize` Number of bytes in the text of the letter. The function ignores this field if you set the `sendAs` parameter to `kSMPSendFileOnlyMask`.

`textBuffer` A pointer to the buffer that contains the text of the letter. The function ignores this field if you set the `sendAs` field to `kSMPSendFileOnlyMask`.

`sendAs`	A constant that indicates whether to send the message as an image (`kSMPSendAsImageMask`), to send the message as a letter with enclosures (`kSMPSendAsEnclosureMask`), to send an enclosed file so that it appears in the In Tray as the file itself rather than as a letter (`kSMPSendFileOnlyMask`), or to send some combination of these formats. You cannot combine the send-file-only and send-as-enclosure formats.
`enclosures`	A pointer to a linked list of enclosure descriptors. If you specify `kSMPSendFileOnlyMask` for the `sendAs` field, you can include only one enclosure. In this case, the enclosure descriptor must provide values for the file creator and type that are appropriate for the file being sent in order for the Finder to display the file correctly.
`drawImageProc`	A pointer to your image-drawing routine. If you want to send a letter as an image, you must provide a routine to draw the image. The procedure declaration for this routine is described on page 3-123. The function ignores this field if you do not set the `sendAs` field to send the file as an image.
`imageRefCon`	A reference constant for your use. The function passes this value to your image-drawing routine.
`supportsColor`	A Boolean value that indicates whether the procedure pointed to by the `drawImageProc` parameter is capable of drawing in color. The Standard Mail Package provides a color graphics port to your image-drawing routine only if you specify `true` for the `supportsColor` field and the user has color QuickDraw.

DESCRIPTION

The `SMPSendLetter` function provides no user interface. Your application must determine the subject, text, enclosures, and addressees for the letter either by providing its own user interface or through some other means. You can use the `SDPGetDirectories`, `SDPFindRecord`, `SDPNewPanel`, or `SDPGetNewPanel` functions to provide a user interface for selecting an addressee.

If the `SMPSendLetter` function returns with a result code that indicates a bad recipient descriptor or a bad enclosure descriptor, you can check the `result` field of each descriptor in the linked list to determine which one was bad. Look in the `filename` field of the bad enclosure descriptor for the name of the file that caused the problem. The `theRID` field of the recipient descriptor contains the record ID containing the name of the addressee. For example, an `RStringPtr` structure pointing to the name of the addressee represented by the first recipient descriptor of the Recipients list is located in `theLetter->toList->theRID.local.name`.

You cannot specify the values `smSystemScript` or `smCurrentScript` for the `script` parameter. To obtain the system script, call the `GetScriptManagerVariable` function with a selector of `smSysScript`. To obtain the current script, call the `FontScript` function.

The `SMPSendLetter` function can send a letter as a note with optional enclosures, as an image of the note and enclosures, as the document file alone, or as some combination of

these formats. Use one or a combination of the following constants in the `sendAs` field to specify the format for the letter:

```
enum {
   kSMPSendAsEnclosureBit, /* appears as letter with enclosures */
   kSMPSendFileOnlyBit,    /* appears as a file in mailbox. */
   kSMPSendAsImageBit      /* letter includes image of content */
};

/* values of SMPPSendAs */
enum {
   kSMPSendAsEnclosureMask = 1<<kSMPSendAsEnclosureBit,
   kSMPSendFileOnlyMask    = 1<<kSMPSendFileOnlyBit,
   kSMPSendAsImageMask     = 1<<kSMPSendAsImageBit
};

typedef Byte SMPPSendAs;
```

Constant descriptions

`kSMPSendAsEnclosureMask`
: The `SMPSendLetter` function sends the letter as a note with the text pointed to by the `textBuffer` parameter and the enclosure specified by the enclosure descriptor.

`kSMPSendFileOnlyMask`
: The enclosed file appears directly in the recipient's In Tray as the file itself rather than as a letter with an enclosure. If you specify this value for the `sendAs` parameter, the letter can contain only one enclosure.

`kSMPSendAsImageMask`
: The `SMPSendLetter` function converts the note into an image and calls your image-drawing routine to convert the enclosures into an image.

To combine formats, perform a bitwise OR operation on the appropriate constants. For example, to send a document as both a note with enclosures and as an image, set the `sendAs` parameter to `kSMPSendAsEnclosureMask PLUS kSMPSendAsImageMask` in Pascal or `kSMPSendAsEnclosureMask OR kSMPSendAsImageMask` in assembly language or C. You cannot combine the send-as-file format (`kSMPSendFileOnlyMask`) with the note-with-enclosures format (`kSMPSendAsEnclosureMask`).

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

You cannot combine the document-only format (`kSMPSendFileOnlyMask`) with the note-with-enclosures format (`kSMPSendAsEnclosureMask`). If you attempt to do so, the function returns the `paramErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$01F4

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in a parameter value

SEE ALSO

The procedure declaration for your image-drawing routine is described on page 3-123.

The enclosure descriptor is defined in "Enclosure Descriptor" on page 3-26.

The recipient descriptor is defined in "Recipient Descriptor" on page 3-25.

The `StGetScriptManagerVariable` function and `FontScript` function are described in *Inside Macintosh: Text* in the chapter "Script Manager."

You can obtain record IDs for the recipients of the letter by using the dialog boxes or the Catalog-Browsing panel described in the chapter "Standard Catalog Package" in this book.

You can create a linked list of record descriptors from the recipient record IDs by calling the `SMPResolveToRecipient` function described on page 3-44.

SMPNewPage

The `SMPNewPage` function creates a new page for use by your image-drawing routine.

```
pascal OSErr SMPNewPage(OpenCPicParams *newHeader);
```

`newHeader` Pointer to an `OpenCPicParams` structure (see the chapter "Color QuickDraw" in *Inside Macintosh: Imaging With QuickDraw*). The `SMPNewPage` function sets the size of your graphics port rectangle equal to the size of the source rectangle you specify in this structure, and sets the image's horizontal and vertical resolutions to those you specify in this structure. For the normal resolution of the Macintosh screen, use 72 pixels per inch for both the vertical and horizontal resolutions.

DESCRIPTION

The `SMPSendLetter` or `SMPImage` function calls your image-drawing routine when you add an image to a letter you are sending. Your image-drawing routine then calls the `SMPNewPage` function before it draws each new page of an image file.

Note
You use the `hRes` and `vRes` fields in the `OpenCPicParams` structure to specify the horizontal and vertical resolutions of the image. Both of these fields are of type `Fixed`, which is a long word that contains an integer part in the high-order word and a binary fraction in the low-order word. To set the horizontal resolution to 72 dpi, for example, you specify a value of 0x00480000 for the `hRes` field to indicate an integer part with a value of 72 and no fractional part. If by mistake you simply specified a value of 72 (that is, 0x00000048) for the `hRes` field, you would be indicating an integer part with a value of 0 and a fractional part of 9/8192. Note also that you can use the `FixRatio` routine to create a value of type `Fixed` from two integer values representing a numerator and denominator. ◆

SPECIAL CONSIDERATIONS

If you change the graphics port within your image-drawing routine, you must change it back before calling the `SMPNewPage` function.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0834

RESULT CODES

`noErr`	0	No error
`kSMPTooManyPages`	–1927	Image is more than 127 pages

SEE ALSO

The `SMPSendLetter` function is described on page 3-37.

The procedure declaration for your image-drawing routine is described on page 3-123.

The `OpenCPicParams` structure is described in the chapter "Color QuickDraw" in *Inside Macintosh: Imaging With QuickDraw*. The `FixRatio` routine is described in *Inside Macintosh: Operating System Utilities*.

SMPImageErr

The `SMPImageErr` function returns result codes from image-drawing routines.

```
pascal OSErr SMPImageErr(void);
```

DESCRIPTION

The `SMPSendLetter` or `SMPImage` function calls your image-drawing routine when you add an image to a letter you are sending. Your image-drawing routine calls the `SMPImageErr` function instead of calling the `QDError` function after it calls each QuickDraw routine. The `SMPImageErr` function returns both QuickDraw errors and errors returned by the `SMPAddBlock` function.

SPECIAL CONSIDERATIONS

If you change the graphics port within your image-drawing routine, you must change it back before calling the `SMPImageErr` function.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0000	$0835

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	Disk full
`pixmapTooDeepErr`	–148	Pixel map structure is deeper than 1 bit per pixel
`mfStackErr`	–149	Insufficient stack
`rgnTooBigErr`	–500	Bitmap would convert to a region greater than 64 KB

SEE ALSO

The `SMPSendLetter` function is described on page 3-37.

The procedure declaration for your image-drawing routine is described on page 3-123.

The `QDError` function is described in the chapter "Color QuickDraw" in *Inside Macintosh: Imaging With QuickDraw*.

The `SMPImageErr` function returns both QuickDraw errors and errors returned by the `SMPAddBlock` function (page 3-91).

SMPResolveToRecipient

The `SMPResolveToRecipient` function takes a pointer to a `PackedDSSpec` structure and returns a pointer to a linked list of mail addresses.

```
pascal OSErr SMPResolveToRecipient(PackedDSSpecPtr dsSpec,
                        SMPRecipientDescriptorPtr *recipientList,
                        AuthIdentity identity);
```

`dsSpec` A pointer to a `PackedDSSpec` structure containing the record ID and location information for a user record or group record.

`recipientList`
A pointer to a linked list of recipients for a letter. You can use this parameter as input to the `SMPSendLetter` function, or you can use the `recipient` field of the recipient descriptor as input to the `SMPAddAddress` function.

`identity` The authentication identity of the caller. The catalog uses this identity to determine whether the caller has the access privileges necessary to resolve specific mail addresses.

DESCRIPTION

When the user selects a record from one of the standard dialog boxes or from the Catalog-Browsing panel, you can use a pointer to the `PackedDSSpec` structure for that record as input to the `SMPResolveToRecipient` function.

If the `PackedDSSpec` structure holds a single address, the function returns a linked list with only one item. If the record is for a group address (that is, if the type of the record is `Group`) and the record is in a personal catalog, then the function resolves it into a linked list of all the members of the group, including all the members of any personal catalog groups in that group. The function performs this service for group addresses in personal catalogs because the recipient is unlikely to have the same information in his or her personal catalog. The function does not expand groups that are not in personal catalogs, because the recipient is assumed to have access to the catalog server to expand those groups.

You can use the linked list returned by the `SMPResolveToRecipient` function as input to the `SMPSendLetter` function.

SPECIAL CONSIDERATIONS

The `SMPResolveToRecipient` function allocates each recipient descriptor in the current heap. To dispose of a recipient descriptor you must first call the `DisposePtr` function to deallocate the `recipient` field in the recipient descriptor, and then call the `DisposePtr` function again to dispose of the recipient descriptor itself.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$044C

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Out of memory

SEE ALSO

The routines for displaying and obtaining information from standard catalog dialog boxes and the Catalog-Browsing panel are described in the chapter "Standard Catalog Package" in this book.

Recipient descriptors are described in "Recipient Descriptor" on page 3-25.

The `SMPSendLetter` function is described on page 3-37. The `SMPAddAddress` function is described on page 3-118.

Providing Mailers in Your Windows

The routines in this section add a mailer to a window and help you to make the mailer appear to be an integral part of your application. You must call the `SMPInitMailer` function before calling any of the other mailer functions.

The `SMPNewMailer` function (page 3-46) adds a new mailer to a window. The `SMPGetDimensions` function (page 3-48) lets you determine the size of a mailer so you can decide how to fit it in your window. You can add a new mailer to the mailer set of a received letter with the `SMPMailerForward` function (page 3-49) or create a new mailer for a reply letter with the `SMPMailerReply` function (page 3-51).

The `SMPExpandOrContract` function (page 3-56) lets you expand or contract a mailer from within your application, and the `SMPMoveMailer` function (page 3-61) lets you move a mailer within your window.

You can use the `SMPGetTabInfo` (page 3-53) and `SMPBecomeTarget` (page 3-54) functions to let the user navigate seamlessly among fields in the mailer and your application window using the Tab key.

You can call the `SMPPrepareToClose` function (page 3-59) to determine whether you can close a window that contains a mailer. You use the `SMPDisposeMailer` function (page 3-61) to remove a mailer from a window and release the memory used by the mailer.

SMPInitMailer

The `SMPInitMailer` function initializes the mailer routines of the Standard Mail Package.

```
pascal OSErr SMPInitMailer(long mailerVersion);
```

`mailerVersion`
The version number of the Standard Mail Package.

DESCRIPTION

You must call this function before the first time you call any other Standard Mail Package function that applies to mailers. If you do not call this function, other mailer functions return the result code `kSMPMailerNotInitialized` when you call them.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1285

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Out of memory

SMPNewMailer

The `SMPNewMailer` function allocates a new mailer for a window you specify.

```
pascal OSErr SMPNewMailer(WindowPtr window,
                          Point upperLeft,
                          Boolean canContract,
                          Boolean initiallyExpanded,
                          AuthIdentity identity,
                          const PrepareMailerForDrawingProcPtr
                                        prepareMailerForDrawingCB,
                          long clientData);
```

`window` The window in which you want the mailer to appear.

`upperLeft` The upper-left corner of the mailer, in the window's local coordinates. This position is normally (0, 0).

`canContract`
A Boolean value that specifies whether it should be possible to contract and expand the mailer. Specify `true` if you want the mailer to have this ability; this parameter should always be set to `true` unless the mailer is in its own, separate window.

`initiallyExpanded`
A Boolean value that specifies whether the mailer is to be expanded or contracted when initially displayed. Specify `true` if you want it to be expanded initially. The function ignores this parameter if the `canContract` parameter is set to `false`.

`identity` The authentication identity of the sender of the letter. Specify 0 to use the identity of the most recently authenticated user. The `SMPNewMailer` function uses the identity to fill in the From field in the mailer.

`prepareMailerForDrawingCB`
A pointer to your drawing-preparation function. If you change the clip region, coordinates, or other aspects of your window's graphics port, you must provide this function to restore the graphics port to a standard state so that the Standard Mail Package can draw a mailer in your window. The drawing-preparation function is described on page 3-122. Specify `nil` for this parameter if you are not providing a drawing-preparation function.

`clientData`
Reserved for your use. The `SMPNewMailer` function passes this value unaltered to your drawing callback routine.

DESCRIPTION

You should call the `SMPNewMailer` function whenever you want to have a mailer appear in a window; for example, when the user chooses the Add Mailer item from the Mail menu in your application. When you call this function, the Standard Mail Package adds a mailer to the window you specify. The next time the user chooses the Save or Save As commands, you should save the document in the letter file format rather than in your application's file format.

If you want the mailer to appear in a modeless, movable dialog box, or for some other reason do not want to provide the user with the ability to expand and contract the mailer, set the `canContract` parameter to `false`.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000C	$125D

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Out of memory
`kSMPMailerNotInitialized`	–1902	The mailer has not been initialized
`kSMPMailerAlreadyInWindow`	–1911	A mailer was previously allocated

SEE ALSO

Use the `SMPBeginSave` function (page 3-77) to save a document in the letter file format.

Use the `SMPDisposeMailer` function (page 3-61) to dispose of a mailer.

Use the `SMPMailerForward` function (page 3-49) to add a mailer to a letter that you want to forward.

Use the `SMPMailerReply` function (page 3-51) to add a reply mailer to a window.

SMPGetDimensions

The `SMPGetDimensions` function returns the standard dimensions of a mailer.

```
pascal OSErr SMPGetDimensions(short *width,
                              short *contractedHeight,
                              short *expandedHeight);
```

`width` A pointer to the minimum width, in QuickDraw coordinates, that bounds all of the fields in a mailer.

`contractedHeight`
A pointer to the height, in QuickDraw coordinates, of a mailer in the contracted state.

`expandedHeight`
A pointer to the height, in QuickDraw coordinates, of a mailer in the expanded state.

DESCRIPTION

The `SMPGetDimensions` function lets you determine the standard dimensions of a mailer from within your program so that your application will continue to work correctly if Apple ever changes the size of a mailer. When the user expands or contracts a mailer, it is up to you to update the content part of your document's window appropriately. You can use the heights returned by the `SMPGetDimensions` function to determine how large an area of your window is affected. You can use the width returned

by this function to help determine the size to make a window when the user clicks the zoom box. Clicking the zoom box should never make a window with a mailer in it smaller than the minimum size of the mailer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$125C

RESULT CODES

`noErr`	0	No error
`kSMPMailerNotInitialized`	–1902	The mailer has not been initialized

SEE ALSO

You use the `SMPMailerEvent` function (page 3-63) to determine when the user has contracted or expanded the mailer.

SMPMailerForward

The `SMPMailerForward` function creates a new mailer for a letter that is to be forwarded.

```
pascal OSErr SMPMailerForward(WindowPtr window,
                              AuthIdentity from);
```

`window` A pointer to the window containing the letter you want to forward.

`from` The authentication identity of the sender of the letter. Specify 0 to use the identity of the user whose mailbox contains the received letter. The `SMPMailerForward` function uses the identity to fill in the From field in the mailer.

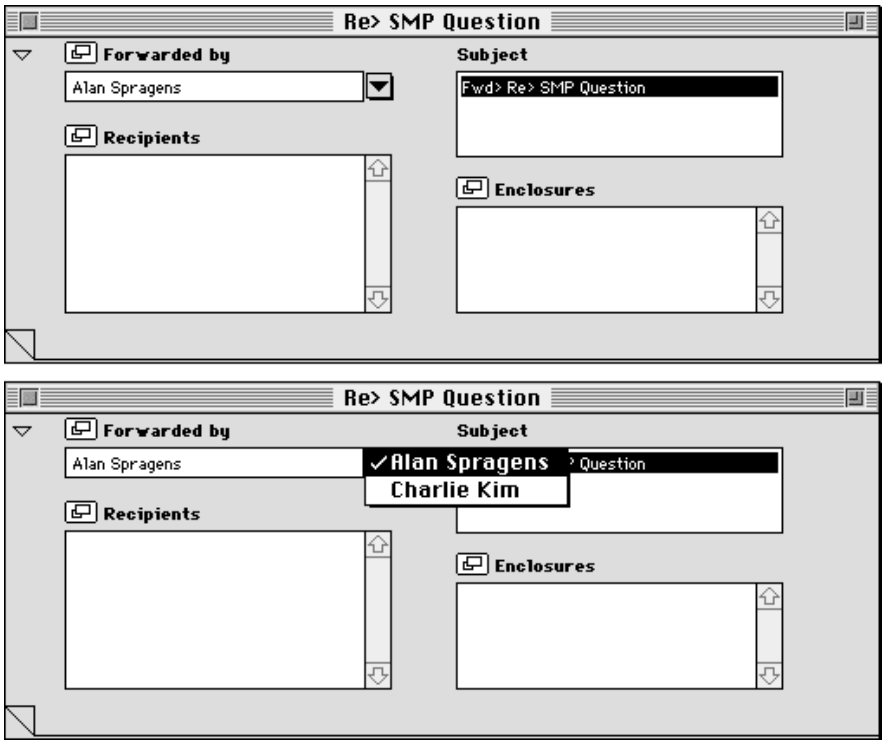
DESCRIPTION

When the user has received a letter and chooses the Forward item in the Mail menu, you call the `SMPMailerForward` function to add a new mailer to the mailer set. The function superimposes the new mailer on the existing mailers in the specified window and, if this is only the second mailer in the mailer set, adds a pop-up menu to the From field in the mailer. If there are already two or more mailers in the mailer set, the function

adds the new mailer to the existing pop-up menu. The user can use this menu to view any of the mailers in the mailer set. Figure 3-6 shows the top mailer and the pop-up menu for a letter that has been forwarded once.

You can call the `SMPMailerForward` function to add a new mailer to a mailer set only if the top mailer in the set is a received mailer. You can use the `hasBeenReceived` field of the `SMPMailerState` structure to get this information.

Figure 3-6 Mailer for a forwarded letter



SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1261

RESULT CODES

noErr	0	No error
kOCEUnknownID	–1567	Authentication identity passed is not valid
kSMPNoMailerInWindow	–1909	No mailer is in the specified window

SEE ALSO

Use the SMPNewMailer function (page 3-46) to add a new mailer to a window that has no mailer.

Use the SMPMailerReply function (described next) to add a new mailer to a letter to which you want to reply.

Use the SMPGetMailerState function (page 3-69) to obtain an SMPMailerState structure. The SMPMailerState structure is described in "Mailer-State Structure" on page 3-30.

SMPMailerReply

The SMPMailerReply function helps you reply to a letter by adding a new mailer to a window you specify and addressing the reply mailer by copying information from the original mailer.

```
pascal OSErr SMPMailerReply(WindowPtr originalLetter,
                            WindowPtr newLetter,
                            Boolean replyToAll,
                            Point upperLeft,
                            Boolean canContract,
                            Boolean initiallyExpanded,
                            AuthIdentity identity,
                            const PrepareMailerForDrawingProcPtr
                                        prepareMailerForDrawingCB,
                            long clientData);
```

originalLetter
A pointer to the window containing the mailer for the original letter to which the user wishes to reply.

newLetter A pointer to the window that you are providing for the reply. The function adds a mailer to this window; the window must not already contain a mailer.

replyToAll
A Boolean value that indicates whether all the original "To" and "cc" recipients should be included as addressees for the reply.

upperLeft The upper-left corner of the mailer in the window's local coordinates. This position is normally (0, 0).

`canContract`
A Boolean value that specifies whether it should be possible to contract and expand the mailer. Specify `true` if you want the mailer to have this ability; this parameter should always be set to `true` unless the mailer is in its own, separate window.

`initiallyExpanded`
A Boolean value that specifies whether the mailer is to be expanded or contracted when initially displayed. Specify `true` if you want it to be expanded initially. The function ignores this parameter if the `canContract` parameter is set to `false`.

`identity` The authentication identity of the sender of the letter. Specify 0 to use the identity of the user whose mailbox contains the received letter. The `SMPMailerReply` function uses the identity to fill in the From field in the mailer.

`prepareMailerForDrawingCB`
A pointer to your drawing-preparation function. If you change the clip region, coordinates, or other aspects of your window's graphics port, you must provide this function to restore the graphics port to a standard state so that the Standard Mail Package can draw a mailer in your window. The drawing-preparation function is described on page 3-122. Specify `nil` for this field if you are not providing a drawing-preparation function.

`clientData`
Reserved for your use. The `SMPMailerReply` function passes this value unaltered to your callback routine.

DESCRIPTION

When the user chooses the Reply or Reply to All items in the Mail menu, you should create a new document window and call the `SMPMailerReply` function. This function places a mailer in the window, copies the subject from the original letter, places the string "Re>" in front of it, and places it in the Subject field of the new mailer. Then it copies the From address from the original letter and places it in the Recipients field of the reply mailer. If the user chose the Reply to All item, you should set the parameter `replyToAll` to `true`, and the `SMPMailerReply` function also copies all the recipients in the Recipients field— including the "cc" recipients—of the received mailer and places them in the corresponding fields of the reply mailer.

If the original letter has been forwarded, the `SMPMailerReply` function takes the subject and addresses from the original letter's most recent mailer; that is, from the mailer that was added the last time the letter was forwarded.

You should call the `SMPMailerReply` function only if the top mailer in the mailer set is a received mailer. You can use the `hasBeenReceived` field of the `SMPMailerState` structure to get this information.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000F	$1262

RESULT CODES

`noErr`	0	No error
`kOCEUnknownID`	–1567	Authentication identity is not valid
`kSMPNoMailerInWindow`	–1909	No mailer is in the specified window
`kSMPMailerAlreadyInWindow`	–1911	Specified window already has a mailer

SEE ALSO

Use the `SMPNewMailer` function (page 3-46) to add a new mailer to a window that has no mailer.

Use the `SMPMailerForward` function (page 3-49) to add a mailer to a letter that you want to forward.

Use the `SMPGetMailerState` function (page 3-69) to obtain an `SMPMailerState` structure. The `SMPMailerState` structure is described in "Mailer-State Structure" on page 3-30.

SMPGetTabInfo

The `SMPGetTabInfo` function tells you which fields in the mailer are the first and last to be highlighted when the user presses the Tab key repeatedly to move from one field to another.

```
pascal OSErr SMPGetTabInfo(SMPMailerComponent *firstTab,
                           SMPMailerComponent *lastTab);
```

`firstTab` The first field highlighted.

`lastTab` The last field highlighted.

DESCRIPTION

When the user first clicks in a mailer, the Standard Mail Package makes one field the target for user actions and highlights that field. If the user presses the Tab key, the Standard Mail Package makes another field the target, and so on, eventually returning to the first field. You can intercept this sequence and make a field in your window active when the user presses the Tab key, returning to the mailer after you have given the user the opportunity to modify one or more fields in your window. The `SMPGetTabInfo` function tells you which field is the first one to be made a target by the Standard Mail Package and which is the last, so you know where to intercept the sequence and where

to return to it. This sequence is not dependent on the state of the mailer; you need call it only once.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1274

RESULT CODES

`noErr`	0	No error
`kSMPMailerNotInitialized`	–1902	Mailer has not been initialized

SEE ALSO

The possible values for the `SMPMailerComponent` data type are shown on page 3-32.

You use the `SMPBecomeTarget` function (described next) to return the Tab sequence to the mailer.

You use the `SMPGetMailerState` function (page 3-69) to determine which field is currently the target.

SMPBecomeTarget

The `SMPBecomeTarget` function specifies whether your window or the mailer is the target of user action and, if the mailer is the target, specifies which field in the mailer is active.

```
pascal OSErr SMPBecomeTarget(WindowPtr window,
                             Boolean becomeTarget,
                             SMPMailerComponent whichField);
```

`window` The window containing the mailer.

`becomeTarget`
A Boolean value that specifies whether the mailer in this window should become the target of the user's actions. If this parameter is set to `true`, the mailer becomes the target. If it is set to `false`, the Standard Mail Package does not highlight any field, and the `SMPMailerEvent` function assumes that key-down events are intended for your application.

`whichField`

If the `becomeTarget` parameter is set to `true`, this parameter specifies which field should be active. If the `becomeTarget` parameter is set to `false`, the function ignores this field. Possible values for this field are shown on page 3-32.

DESCRIPTION

The user can use the Tab key to cycle through the fields in the mailer. Each time the user presses the Tab key, you receive a key-down event. In most cases, you would call the `SMPMailerEvent` function to handle the event. However, if you want one or more fields in your application's window to be included in the set of fields that the user can select with the Tab key, you must determine the nature of the key-down event yourself. If the user pressed the Tab key, you can call the `SMPGetMailerState` function (page 3-69) to determine which field in the mailer is currently the target. You can then check the results of the `SMPGetTabInfo` function to find out which field is the last one in the sequence. If the current field is the one returned in the `lastTab` parameter of the `SMPGetTabInfo` function, you can call the `SMPBecomeTarget` function with the `becomeTarget` parameter set to `false`. You can then activate and highlight whichever field in your window you wish.

When you have finished cycling through the fields in your window, call the `SMPBecomeTarget` function again, this time with the `becomeTarget` parameter set to `true` and the `whichField` parameter set to the value returned in the `firstTab` parameter of the `SMPGetTabInfo` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0005	$1273

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in this window
`kSMPIllegalComponent`	–1918	Bad field name parameter
`kSMPMailerAlreadyNotTarget`	–1919	This mailer is not the target
`kSMPComponentIsAlreadyTarget`	–1920	The selected field is the target

SEE ALSO

The possible values for the `SMPMailerComponent` data type are shown on page 3-32.

You can call the `SMPGetMailerState` function (page 3-69) to determine which field in the mailer is currently the target.

You can call the `SMPGetTabInfo` function (page 3-53) to find out which fields are the first and last in the selection sequence.

You can call the `SMPMailerEvent` function (page 3-63) each time you receive an event. This function returns a value telling you how the Standard Mail Package handled the event and whether your application has to process it as well.

SMPExpandOrContract

The `SMPExpandOrContract` function expands or contracts a mailer.

```
pascal OSErr SMPExpandOrContract(WindowPtr window,
                                 Boolean expand);
```

`window` The window containing the mailer.

`expand` A Boolean value that specifies whether the mailer in this window should be expanded (`true`) or contracted (`false`).

DESCRIPTION

The user indicates a desire to expand or contract a mailer by clicking the triangle in the upper-left corner of the mailer (see Figure 3-2 and Figure 3-3 on page 3-5). If the user wants to expand the mailer, the `SMPMailerEvent` function returns the flag `kExpanded`. You must update your window to make room for the expanded mailer and then call the `SMPExpandOrContract` function to expand the mailer. (When the user contracts the mailer, by contrast, you have to update the content portion of your window but do not have to call the `SMPExpandOrContract` function.)

The `SMPExpandOrContract` function also lets you expand or contract the mailer entirely from within your application, for example, to implement an Expand or Contract menu command.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$1272

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPMailerCannotExpandOrContract`	–1916	Mailer created with `canContract` parameter set to `false`
`kSMPMailerAlreadyExpandedOrContracted`	–1917	Mailer is already in requested state

SEE ALSO

You set the initial state of the mailer (expanded or contracted) with the `SMPNewMailer` function (page 3-46), the `SMPMailerReply` function (page 3-51), or the `SMPOpenLetter` function (page 3-94).

You can call the `SMPGetMailerState` function (page 3-69) to determine whether the mailer is currently expanded or contracted.

You call the `SMPGetDimensions` function (page 3-48) to determine the size of an expanded or contracted mailer.

SMPMoveMailer

The `SMPMoveMailer` function moves a mailer within your window.

```
pascal OSErr SMPMoveMailer(WindowPtr window,
                           short dh,
                           short dv);
```

`window`	The window containing the mailer you want to move.
`dh`	The horizontal distance, in QuickDraw coordinates, by which you want to move the mailer. Use a positive number to move the mailer to the right and a negative number to move the mailer to the left.
`dv`	The vertical distance, in QuickDraw coordinates, by which you want to move the mailer. Use a positive number to move the mailer down and a negative number to move the mailer up.

DESCRIPTION

You set the initial location of a mailer in your window when you call the `SMPNewMailer` function or the `SMPMailerReply` function. You can use the `SMPMoveMailer` function to move a mailer if, for example, you need to make space for a tool palette at the top or left edge of your window.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$126A

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You set the initial location of the mailer with the `SMPNewMailer` function (page 3-46), the `SMPMailerReply` function (page 3-51), or the `SMPOpenLetter` function (page 3-94).

SMPTagDialog

The `SMPTagDialog` function displays a dialog box that allows a user to add a tag to a letter that was opened from the mailbox.

```
pascal OSErr SMPTagDialog(WindowPtr window,
                          RString32 *theTag);
```

`window` The window containing the mailer.

`theTag` A pointer to the tag to be associated with the letter. If you specify a tag when you call the function, it is displayed as the default value in the dialog box. The function uses this parameter to return the tag specified by the user.

DESCRIPTION

The PowerTalk mailbox allows the user to sort and display letters according to tags that the user has specified for each letter. Your application can provide a Tag item in the Mail menu. If the user chooses this item, you should call the `SMPTagDialog` function to let the user specify the tag. You should call the `SMPGetMailerState` function to determine whether to enable the Tag command.

The AOCE software stores the tag with the letter and displays it for the user in the In and Out Trays. It is not necessary for you to specify this tag in the close-options structure when you call the `SMPCloseOptionsDialog` or `SMPDisposeMailer` functions. When you save the letter to disk, the letter becomes an HFS object and no longer has a tag.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$128B

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in a parameter value
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

Call the `SMPGetMailerState` function (page 3-69) to determine whether to enable the Tag command.

SMPPrepareToClose

The `SMPPrepareToClose` function tells you whether a mailer can be closed.

```
pascal OSErr SMPPrepareToClose(WindowPtr window);
```

`window` The window containing the mailer that you would like to close.

DESCRIPTION

In certain circumstances—for instance, when an enclosure is open—you can't dispose of a mailer. The `SMPPrepareToClose` function returns an error when you can't dispose of a mailer, so you can display a dialog box informing the user of the situation rather than closing the window containing the mailer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1287

RESULT CODES

`noErr`	0	No error
`kSMPCopyInProgress`	–1901	Finder is copying an enclosure
`kSMPHasOpenAttachments`	–1906	One or more enclosures are open
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You should call the `SMPPrepareToClose` function before closing a window containing a mailer or attempting to call the `SMPDisposeMailer` function (page 3-61).

SMPCloseOptionsDialog

The `SMPCloseOptionsDialog` function displays a dialog box that allows a user to delete letters or add tags to letters that were opened from the mailbox.

```
pascal OSErr SMPCloseOptionsDialog(WindowPtr window,
                                   SMPCloseOptionsPtr closeOptions);
```

`window` The window containing the mailer.

`closeOptions`
A pointer to a close-options structure that specifies the initial settings to be displayed in the dialog box. After the function call returns, this structure contains the new settings entered by the user in the close-options dialog box.

DESCRIPTION

Your application should provide a user preference option that specifies whether attempting to close a letter should cause the close-options dialog box to appear. If the user elects to see the dialog box, you should call the `SMPCloseOptionsDialog` function whenever a user closes a window containing a mailer and before you call the `SMPDisposeMailer` function. If the user opened the letter from the mailbox, the `SMPCloseOptionsDialog` function displays the close-options dialog box; otherwise, the function does nothing.

You can use the `closeOptions` parameter to provide default settings for the dialog box. If you provide a Close and Delete item in the File menu and the user chooses this item, you should specify `true` for the `moveToTrash` field of the structure pointed to by the `closeOptions` parameter.

You can use the `tag` field of the structure pointed to by the `closeOptions` parameter to specify a default tag for the letter. If the letter already has a tag value, either because the user added it the last time the letter was closed or because you called the `SMPTagDialog` function, the Standard Mail Package puts that tag in the `tag` field of the dialog box. It is not necessary for you to specify this tag in the close-options structure when you call the `SMPCloseOptionsDialog` function.

The Standard Mail Package stores the options the user selects and executes them when you call the `SMPDisposeMailer` function (if the `closeOptions` parameter in the `SMPDisposeMailer` function is not set to `nil`).

SPECIAL CONSIDERATIONS

If you specify `true` for both the `moveToTrash` field and the `addTag` field of the close-options structure, the `SMPCloseOptionsDialog` function returns the `paramErr` result code.

If you specify `true` for the `addTag` field of the close-options structure, you must also specify a valid tag for the letter. If you specify `true` for the `addTag` field and you specify a zero-length string for the `tag` field, the function returns the `paramErr` result code.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1288

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in a parameter value
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The close-options structure is described in "Close-Options Structure" on page 3-29.

You can call the `SMPTagDialog` function (page 3-58) to display a dialog box that allows the user to add a tag to a letter in the In Tray.

Call the `SMPDisposeMailer` function (described next) to execute the close options.

SMPDisposeMailer

The `SMPDisposeMailer` function deallocates the mailer set in the specified window and erases the mailer set.

```
pascal OSErr SMPDisposeMailer(WindowPtr window,
                              SMPCloseOptionsPtr closeOptions);
```

`window` The window containing the mailer set you want to deallocate.

`closeOptions`
: A pointer to a close-options structure specifying actions the Standard Mail Package should take in addition to disposing of the mailer set. If you specify nil for this parameter, the function disposes of the mailer set without taking any other action.

DESCRIPTION

You should call the `SMPDisposeMailer` function when the user chooses the Remove Mailer item from a menu or when you close a window that contains a mailer. This function removes the mailer set from the window you specify and deallocates all the data structures associated with that mailer set. If the user removes the mailer from the window, the next time the user chooses the Save or Save As commands, you should save the document in your application's file format rather than the letter file format.

You use the `closeOptions` parameter to specify close options. For example, you can provide a Close and Delete item in the File menu. If the user chooses this item, you should specify `true` for the `moveToTrash` field of the structure pointed to by the `closeOptions` parameter.

Your application may provide a user preference option that specifies whether attempting to close a letter should cause the close-options dialog box to appear. If the user elects to see the dialog box, you should call the `SMPCloseOptionsDialog` function whenever the user closes a window containing a mailer. Then use the pointer to the close-options structure that you provided to the `SMPCloseOptionsDialog` function as the value of the `closeOptions` parameter of the `SMPDisposeMailer` function.

Before you close a window that contains a mailer, call the `SMPPrepareToClose` function to make sure that you can dispose of the mailer.

SPECIAL CONSIDERATIONS

If you specify `true` for both the `moveToTrash` field and the `addTag` field of the close-options structure, the `SMPDisposeMailer` function returns the `paramErr` result code.

If you specify `true` for the `addTag` field of the close-options structure, you must also specify a valid tag for the letter. If you specify `true` for the `addTag` field and you specify a zero-length string for the `tag` field, the function returns the `paramErr` result code.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$125E

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in a parameter value
`kSMPHasOpenAttachments`	–1906	One or more enclosures are open
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The `SMPPrepareToClose` function (page 3-59) tells you whether it's possible to dispose of a mailer.

The close-options structure is described in "Close-Options Structure" on page 3-29.

The `SMPCloseOptionsDialog` function (page 3-60) displays a dialog box that lets the user select close options for a letter that was opened from the mailbox.

Handling Events in Mailers

Whenever you receive an event for a window that contains a mailer, you can pass that event directly to the `SMPMailerEvent` function (described next). The Standard Mail Package handles the event if it applied to the mailer and returns a value that tells you what action it took and whether you have to take any further action. You can also use the `SMPMailerEditCommand` function (page 3-67) to handle events related to standard items in the Edit menu.

When the user is working in the mailer, you must enable and disable items in the Edit menu as appropriate. The `SMPGetMailerState` function (page 3-69) lets you determine which items should be enabled or disabled. You must ensure that the Undo command works consistently whether the user is working in the mailer or in your application. You use the `SMPGetMailerState` function to determine when to clear your application's undo buffer, and you use the `SMPClearUndo` function (page 3-70) to tell the mailer when to clear its Undo buffer.

Finally, you can use the `SMPDrawMailer` function (page 3-72) to redraw the mailer if you want to handle update events yourself or if you need to redraw the mailer for some other reason.

SMPMailerEvent

The `SMPMailerEvent` function processes events that you pass to it, gives you information about how the Standard Mail Package responded to the event, and informs you of further action that you must take.

```
pascal OSErr SMPMailerEvent(const EventRecord *event,
                            SMPMailerResult *whatHappened,
                            const FrontWindowProcPtr frontWindowCB,
                            long clientData);
```

`event` A pointer to the event record of an event returned to your application by the `WaitNextEvent` function.

`whatHappened`
A pointer to a set of flags informing you what action the `SMPMailerEvent` function took.

`frontWindowCB`
A pointer to your front-window routine. This routine, described on page 3-124, returns a pointer to the window that your application wants the Standard Mail Package to consider as the front window. Specify `nil` for this field if you do not want to provide a front-window routine. If you do not provide a front-window routine, the Standard Mail Package uses the Window Manager's `FrontWindow` routine.

`clientData`
Reserved for your use. The `SMPMailerEvent` function passes this value unaltered to your callback routine.

DESCRIPTION

Each time your application calls the `WaitNextEvent` function, it can pass the event record immediately to the `SMPMailerEvent` function. The `SMPMailerEvent` function determines whether the Standard Mail Package should handle the event, your application should handle the event, or action is required by both the Standard Mail Package and your application. If the `SMPMailerEvent` function has to take any further action, it does so before returning control to your application. In any case, the `whatHappened` parameter returns a set of flags that tell you what action, if any, the function took, and whether your application must handle the event.

If the event record does not include a window pointer, the `SMPMailerEvent` function uses your front-window callback routine to determine to which window the event applies. If you do not provide a front-window callback routine, the `SMPMailerEvent` function uses the Window Manager's `FrontWindow` routine.

If you decide instead to check the event record first and pass to the `SMPMailerEvent` function only events that the Standard Mail Package must handle, call the `SMPBecomeTarget` function when the mailer is no longer the target (for example, when the user clicks in the content region of the window). In that case, you must still pass null events to the `SMPMailerEvent` function frequently so that the Standard Mail Package can control the appearance of the cursor, implement Balloon Help, and pass null events to the Catalog-Browsing panel and Find-Record panel.

IMPORTANT

To use the Standard Mail Package, your application must be aware of high-level events. You must pass all high-level events (including Apple events) to the `SMPMailerEvent` function before calling the `AEProcessAppleEvent` or `AcceptHighLevelEvent` routines. If you do not do so, some Standard Mail Package features, such as enclosing files and folders, may not work correctly. ▲

The flags returned by the `whatHappened` parameter are as follows:

```
enum {
   kSMPAppMustHandleEventBit,
   kSMPAppShouldIgnoreEventBit,
   kSMPContractedBit,
   kSMPExpandedBit,
   kSMPMailerBecomesTargetBit,
   kSMPAppBecomesTargetBit,
   kSMPCursorOverMailerBit,
   kSMPCreateCopyWindowBit,
   kSMPDisposeCopyWindowBit
};
```

You can use the following masks to test for these bits:

```
enum {
   kSMPAppMustHandleEventMask    = 1<<kSMPAppMustHandleEventBit,
   kSMPAppShouldIgnoreEventMask  = 1<<kSMPAppShouldIgnoreEventBit,
   kSMPContractedMask            = 1<<kSMPContractedBit,
   kSMPExpandedMask              = 1<<kSMPExpandedBit,
   kSMPMailerBecomesTargetMask   = 1<<kSMPMailerBecomesTargetBit,
   kSMPAppBecomesTargetMask      = 1<<kSMPAppBecomesTargetBit,
   kSMPCursorOverMailerMask      = 1<<kSMPCursorOverMailerBit,
   kSMPCreateCopyWindowMask      = 1<<kSMPCreateCopyWindowBit,
   kSMPDisposeCopyWindowMask     = 1<<kSMPDisposeCopyWindowBit
   };

typedef unsigned long SMPMailerResult;
```

Bit descriptions

`kSMPAppMustHandleEventBit`
: The application must process the event. The event was either an event the Standard Mail Package couldn't process, or it was one that both the application and the Standard Mail Package must process (such as activate and update events). The function always sets either this flag or the `kSMPAppShouldIgnoreEventBit` flag.

`kSMPAppShouldIgnoreEventBit`
: The application should ignore the event. It was handled by the Standard Mail Package. The function always sets either this flag or the `kSMPAppMustHandleEventBit` flag.

`kSMPContractedBit`
The user clicked the triangle at the left edge of the mailer (see Figure 3-3 on page 3-5), switching the mailer to the contracted state. Because your application does not have to read the event record, the function also sets the `kSMPAppShouldIgnoreEventBit` flag. However, you must update the content portion of your application's frontmost window when you receive this flag.

`kSMPExpandedBit`
The user clicked the triangle at the left edge of the mailer (see Figure 3-2 on page 3-5), indicating a desire to switch the mailer to the expanded state. Because your application does not have to read the event record, the function also sets the `kSMPAppShouldIgnoreEventBit` flag. However, you must update the content portion of your application's frontmost window and call the `SMPExpandOrContract` function (page 3-56) to finish expanding the mailer when you receive this flag.

`kSMPMailerBecomesTargetBit`
The user had been working in the application's part of the window but has now clicked in the mailer or contracted the mailer. Because your application does not have to read the event record, the function also sets the `kSMPAppShouldIgnoreEventBit` flag. However, you might want to take other action, such as removing highlighting from the content portion of the window or stopping an insertion-point caret from blinking.

`kSMPAppBecomesTargetBit`
The user had been working in the mailer but has now clicked in the application's part of the window. Because you must handle this event, the function also sets the `kSMPAppMustHandleEventBit` flag.

`kSMPCursorOverMailerBit`
When this flag is set, the cursor is in the mailer in the frontmost window, so the Standard Mail Package is controlling Balloon Help and the appearance of the cursor. When this flag is cleared, the cursor is not in the mailer, so you must control the appearance of the cursor. The function also sets the `kSMPAppMustHandleEventBit` flag when it sets or clears the `kSMPCursorOverMailerBit` flag. Because the Standard Mail Package can detect the position of the cursor only when you give it some processing time, the function sets or clears this flag only when you pass it a null event.

`kSMPCreateCopyWindowBit`
The Standard Mail Package is using the Finder to copy files and has displayed a modal dialog box showing the status of the copy operation. You should continue to send events to the `SMPMailerEvent` function.

`kSMPDisposeCopyWindowBit`
The Standard Mail Package has removed the copy status dialog box. Your application should resume normal operation.

SPECIAL CONSIDERATIONS

The Standard Mail Package reserves all high-level events of class `'cwin'` for its own use. Do not install an event handler for events of this class. (There is no problem if you have installed an Apple event handler with class and ID of `typeWildCard`, because the Standard Mail Package removes that handler before calling the `AEProcessAppleEvent` routine and reinstalls it afterward.)

The `SMPMailerEvent` function may move or purge memory; you should not call this function at interrupt time.

The `SMPMailerEvent` function preserves your application's A5 world when it calls your front-window routines. Therefore, you have access to your application's global variables from these routines.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0008	$125F

RESULT CODES

`noErr`	0	No error

SEE ALSO

The front-window callback function is described on page 3-124.

If you check the event record first and pass to the `SMPMailerEvent` function only events that the Standard Mail Package must handle, call the `SMPBecomeTarget` function (page 3-54) when the mailer is no longer the target.

Call the `SMPGetMailerState` function (page 3-69) to determine the current state of the mailer.

SMPMailerEditCommand

The `SMPMailerEditCommand` function handles Edit menu commands when they apply to fields in the mailer.

```
pascal OSErr SMPMailerEditCommand(WindowPtr window,
                        SMPEditCommand command,
                        SMPMailerResult *whatHappened);
```

`window` A pointer to the window containing the mailer.

`command` The Edit menu command that the user chose.

`whatHappened`
A pointer to a set of flags that indicate what action the function took and whether you must take any action. This function sets either the `kSMPAppMustHandleEventBit` or the `kSMPAppShouldIgnoreEventBit` flag.

DESCRIPTION

When the user chooses one of the standard Edit menu commands (Undo, Cut, Copy, Paste, Clear, or Select All), you can call the `SMPMailerEditCommand` function immediately. If the user has selected something in the mailer, the `SMPMailerEditCommand` function handles the requested action, sets the `kSMPAppShouldIgnoreEventBit` flag in the `whatHappened` parameter, and returns. If the user has not selected anything in the mailer, the function sets the `kSMPAppMustHandleEventBit` flag and returns to you immediately. Use the `SMPGetMailerState` function to determine which of the Edit menu commands to enable.

The possible values for the `command` parameter are as follows:

```
enum {
   kSMPUndoCommand,
   kSMPCutCommand,
   kSMPCopyCommand,
   kSMPPasteCommand,
   kSMPClearCommand,
   kSMPSelectAllCommand
};

typedef unsigned short SMPEditCommand;
```

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0005	$1260

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPIllegalComponent`	–1918	Bad `command` parameter value

SEE ALSO

The `flag` field for the `whatHappened` parameter is completely defined on page 3-65.

You can use the `SMPGetMailerState` function (page 3-69) to determine whether the user is working in the mailer and, if so, which of the Edit menu commands to enable.

You can call the `SMPMailerEvent` function (page 3-63) each time you receive an event. This function returns a value telling you how the Standard Mail Package handled the event and whether your application has to process it as well. The `SMPMailerEvent` function returns the value `kSMPAppMustHandleEventBit` when the event is an Edit menu command.

SMPGetMailerState

The `SMPGetMailerState` function returns the state of the specified mailer.

```
pascal OSErr SMPGetMailerState(windowPtr window,
                        SMPMailerState *itsState);
```

`window`	The window containing the mailer whose state you want to know.
`itsState`	A pointer to a structure containing the state of the mailer. The `SMPMailerState` data type is defined and all of its fields are described in "Mailer-State Structure" on page 3-30.

DESCRIPTION

The `SMPGetMailerState` function lets you determine whether the user is working in the mailer, and if so, which Edit menu and Mail menu commands you should enable. For example, if both the `isTarget` and `canCut` fields are set to `true`, then you should enable the Cut item in the Edit menu. This function also returns a value that helps you determine whether to clear your application's undo buffer. You should call this function when you need to display the Edit menu (that is, before calling the `MenuSelect` function) or the Mail menu or when the user presses a keyboard equivalent for an Edit menu command or Mail menu command.

This function also returns other information about the mailer, such as its current state (contracted or expanded), its location in the window, and the number of mailers in the mailer set.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1263

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The `SMPMailerState` data type is defined and all of its fields are described in "Mailer-State Structure" on page 3-30.

You can call the `SMPMailerEvent` function (page 3-63) each time you receive an event. This function returns a value telling you how the Standard Mail Package handled the event and whether your application has to process it as well.

SMPClearUndo

The `SMPClearUndo` function tells the Standard Mail Package to clear its undo buffer.

```
pascal OSErr SMPClearUndo(WindowPtr window);
```

`window` A pointer to the window containing the mailer.

DESCRIPTION

The *Macintosh Human Interface Guidelines* call for an Undo item in the Edit menu and specify that only the latest action can be undone. Furthermore, certain actions that cannot be undone should cause you to disable the Undo item and some should not; for example, you should disable the Undo item after the user saves a file but not after the user scrolls through the window. Even though the Standard Mail Package maintains its own undo buffer, you are responsible for enabling and disabling the Undo item in the Edit menu whether the user is working in the content portion of your window or in the mailer. The `SMPClearUndo` function lets you coordinate the Standard Mail Package's undo facility with that of your application so that it appears to the user that there is only one undo buffer for the entire window.

You should call the `SMPGetMailerState` function when you need to display the Edit menu (that is, before calling the `MenuSelect` routine) or when the user presses a keyboard equivalent for an Edit menu command. If the `SMPGetMailerState` function indicates that the user is working in the content portion of the window, you must determine whether the user can undo the action, and you must enable or disable the Undo item in the Edit menu accordingly. If the action can be undone or if it causes you to disable the Undo item, you must call the `SMPClearUndo` function to tell the Standard Mail Package to clear its undo buffer. If you fail to do so, the Standard Mail Package will not update the mailer state correctly.

Conversely, if the user's last action was in the mailer, you must call the `SMPGetMailerState` function to find out whether to enable or disable the Undo item in the Edit menu and whether to clear your application's undo buffer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1275

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You can call the `SMPMailerEvent` function (page 3-63) each time you receive an event. This function returns a value telling you how the Standard Mail Package handled the event and whether your application has to process it as well.

The `SMPGetMailerState` function (page 3-69) returns a mailer-state structure; see "Mailer-State Structure" on page 3-30. The `undoState` field of the mailer-state structure, described on page 3-33, returns a value that tells you whether to clear your application's undo buffer and whether to disable the Undo item in the Edit menu.

You can use the `SMPMailerEditCommand` function (page 3-67) to handle edit commands when they apply to the mailer.

SMPDrawMailer

The `SMPDrawMailer` function redraws the mailer in the window you specify.

```
pascal OSErr SMPDrawMailer(WindowPtr window);
```

`window` A pointer to the window containing the mailer you want to draw.

DESCRIPTION

You use the `SMPDrawMailer` function to redraw a mailer when you need to do so but have not received an update event, or if you want to handle an update event yourself rather than passing it to the `SMPMailerEvent` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1269

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You can have the Standard Mail Package draw the mailer by passing update events to the `SMPMailerEvent` function (page 3-63).

Sending and Saving Mail

The `SMPBeginSend` function (page 3-81) starts the process of creating a letter that is to be mailed. Immediately before you call the `SMPBeginSend` function, you should call the `SMPSendOptionsDialog` function (page 3-73) to let the user set send options.

The `SMPSendOptionsDialog` function returns the user's choice: whether to send the letter as an image, as standard interchange format, in one of your application's native formats, or in some combination of these three options. To send the letter as an image, you must call the `SMPImage` function (page 3-88) to put an image block in the letter. Call the `SMPAddContent` function (page 3-85) to add a standard interchange format block to the letter and the `SMPAddMainEnclosure` function (page 3-90) to add a document in

one of its native formats to the letter. You can also call the `SMPAddBlock` function (page 3-91) to add one or more blocks of your own design to the letter.

You can enclose files or folders in a letter by calling the `SMPAddAttachment` function (page 3-119) or the `SMPAttachDialog` function (page 3-119).

When you are ready to send the letter, call the `SMPEndSend` function (page 3-84).

The process of saving a letter to disk is similar to the process of sending one. First you call the `SMPBeginSave` function (page 3-77) to create the letter. Then you can use the `SMPAddContent`, `SMPAddMainEnclosure`, and `SMPAddBlock` functions to add content to the letter. To save the letter, call the `SMPEndSave` function (page 3-80). You can use the `SMPOpenLetter` function (page 3-94) to open a letter on disk for reading.

SMPSendOptionsDialog

The `SMPSendOptionsDialog` function displays the send-options dialog box and returns the user's selections.

```
pascal OSErr SMPSendOptionsDialog(WindowPtr window,
                                  Str255 documentName,
                                  StringPtr nativeFormatNames[],
                                  unsigned short nameCount,
                                  SMPSendFormatMask canSend,
                                  SMPSendFormat *currentFormat,
                                  SendOptionsFilterProc filterProc,
                                  long clientData,
                                  SMPSendFormat *shouldSend,
                                  SMPSendOptionsPtr sendOptions);
```

`window` A pointer to the window containing the mailer.

`documentName`
The name of the document. This name is displayed at the top of the send-options dialog box.

`nativeFormatNames`
An array of string pointers containing the names of the "native" formats your application can use for the letter. These names should be the same as the formats listed in the dialog box you display when the user chooses Save As from the File menu. If your application can write data in only one format, use the name of the application. The Standard Mail Package displays the names you list here in a pop-up menu in the send-options dialog box.

`nameCount` The number of string pointers in the `nativeFormatNames` parameter.

`canSend` A set of flags indicating which types of format your application can send for this letter. You can use any combination of the mask values `kSMPNativeMask`, `kSMPImageMask`, and `kSMPStandardInterchangeMask`.

`currentFormat`
A pointer to a send-format structure. If the user opened this letter from the mail box or from disk, you should use this structure to indicate which formats the letter currently contains. If this is a new letter, you should indicate which formats you prefer to use for this letter. Do not include any format you did not include in the canSend parameter.

`filterProc`
A pointer to a routine you can provide to add additional items to the send-options dialog box. This routine is described on page 3-125.

`clientData`
A constant reserved for your use. The Standard Mail Package passes this value to the routine you provide in the `filterProc` parameter.

`shouldSend`
A pointer to a send-format structure, allocated by your application, in which the `SMPSendOptionsDialog` function returns the formats the user has selected for sending the letter.

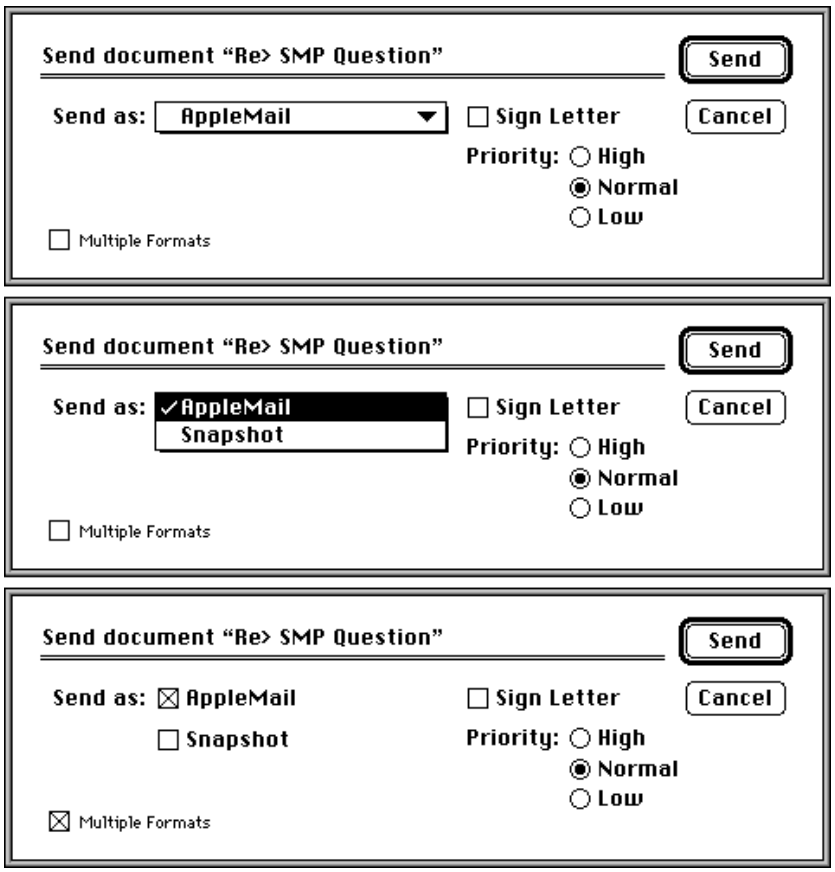
`sendOptions`
A pointer to a send-options structure. Pass this pointer to the `SMPBeginSend` function when you are ready to send the letter. The function only returns information in this structure; it ignores any values in this structure that are set at the time you call the function.

DESCRIPTION

The send-options dialog box lets the user specify whether letters should be signed and whether documents should be sent as application documents, images, or both.

Figure 3-7 shows a send-options dialog box.

Figure 3-7 Send-options dialog box



To make it possible for any user to read letters sent by your application, even if the recipient doesn't have your application, you should be able to add either a standard interchange format version of your document, an image version, or both to the letter. If the standard interchange format does not completely describe your application's documents, you can also send a document in one of its native formats either as a main enclosure to the letter or as a block or blocks that you add to the letter. You use the `canSend` parameter to specify which formats you are prepared to add to a letter.

You should call the `SMPSendOptionsDialog` function before you use the `SMPBeginSend` function to initiate the process of sending a letter. The `SMPBeginSend` function returns a send-format structure that indicates whether the user wants to send an image, standard interchange format, one of the document formats supported by your application, or some combination of the three. If the reader wants to send an image, you must call the `SMPImage` function so that the Standard Mail Package can provide the image. If the reader wants to send standard interchange format, call the `SMPAddContent` function. If the user wants to send the letter as a document, the send-format structure also indicates which of your application's document formats to use.

To add your own items to the send-options dialog box, provide a send-options filter routine.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

The `SMPSendOptionsDialog` function only returns the user's selections. You cannot use this function to set default values for the fields in the send-options dialog box.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0013	$1388

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in a parameter value
`userCanceledErr`	–128	User clicked Cancel button
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPIllegalSendFormats`	–1923	Format not in `canSend` parameter

SEE ALSO

See the descriptions of the `SMPBeginSend` (page 3-81) and `SMPEndSend` (page 3-84) functions for more information about sending mail.

The send-format structure is defined in "Send-Format Structure" on page 3-34. The send-options structure is defined in "Send-Options Structure" on page 3-34.

You can specify a routine to add items to the send-options dialog box. See the description of the `MySendOptionsFilterProc` routine on page 3-125 for more information.

Call the `SMPAddMainEnclosure` function (page 3-90) to add a main enclosure to a letter. Call the `SMPAddContent` function (page 3-85) to add standard interchange format content to a letter. Call the `SMPImage` function (page 3-88) to add an image to the letter.

SMPContentChanged

The `SMPContentChanged` function informs the Standard Mail Package that the content of the letter has changed.

```
pascal OSErr SMPContentChanged(WindowPtr window);
```

`window` A pointer to the window containing the mailer.

DESCRIPTION

You must call the `SMPContentChanged` function to inform the Standard Mail Package when the user changes the content of a letter. The Standard Mail Package can then indicate to the user that the signature is not valid. The Standard Mail Package also needs this information to determine whether it can save or send a forwarded letter without requiring you to rebuild the content of the letter.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$126F

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

To forward a letter, see the `SMPMailerForward` function (page 3-49).

To save a letter, see the `SMPBeginSave` function (page 3-77). To send a letter, see the `SMPBeginSend` function (page 3-81).

Before you allow the user to change the content of a letter, call the `SMPPrepareToChange` function (page 3-83).

SMPBeginSave

You must call the `SMPBeginSave` function before you save a letter.

```
pascal OSErr SMPBeginSave(WindowPtr window,
                          const FSSpec *diskLetter,
                          OSType creator,
                          OSType filetype,
                          SMPSaveType saveType,
                          Boolean *mustAddContent);
```

`window` The window containing the letter to be saved.

`diskLetter`
A pointer to a file system specification structure indicating the name and location you want to use for the file.

`creator` The creator for the file, which also becomes the letter's creator. Use the same creator that you use for all of your application's documents.

`filetype` The file type, which also becomes the letter type. Letters containing only AOCE standard content should be of type `'lttr'`.

`saveType` The type of save: `kSMPSave`, `kSMPSaveAs`, or `kSMPSaveACopy.`

`mustAddContent`
A pointer to a Boolean value returned by the function that tells you whether you have to add any blocks or enclosures to the letter. If this parameter is set to `false`, call the `SMPEndSave` function immediately without adding blocks or enclosures to the letter.

DESCRIPTION

When you save a document to which you have added a mailer, you must save it in letter file format rather than the document format normally used by your application. A letter consists of a header, data blocks, and enclosures. Every block has a block creator and type, and every letter has a letter creator and type. When you save the letter, the Standard Mail Package assigns the file the same creator and type as the letter. Your application should provide icon resources and a file reference resource for your application's letters so that users can distinguish them from standard documents.

To begin the process of saving a letter, you call the `SMPBeginSave` function. This function prepares the letter file into which you can save your document. You can create a new file format for your application's documents that takes advantage of the block structure of a letter file, or you can save your document in one of your application's native formats to a temporary file on disk and then add that file as the main enclosure to the letter. The letter is not actually saved to disk until you call the `SMPEndSave` function.

If neither your application nor the user has changed the content of a received letter, the `SMPBeginSave` function returns a value of `false` for the `mustAddContent` parameter. In that case you can call the `SMPEndSave` function immediately to save the letter. If you have changed the content of the letter, however, the `mustAddContent` parameter returns `true` and you must build the letter (adding the appropriate combination of blocks, main enclosure, standard interchange format block, and image block) just as if it were a new letter. The Standard Mail Package handles enclosures added by the user. Note that, if you make changes to the enclosed original letter, you invalidate any digital signature.

IMPORTANT
Be sure to call the `SMPContentChanged` function whenever the user changes the content of a letter. If you do not call the `SMPContentChanged` function, then the `SMPBeginSave` function doesn't know that the letter has been changed and won't return `true` as the value of `mustAddContent`. ▲

Note
The `SMPBeginSave` and `SMPEndSave` function pair perform a "safe save"; that is, they save the document into a temporary file and then, if the document has been saved before, replace the original file with the temporary file. ◆

When you call the `SMPBeginSave` function you must specify the type of save, as follows:

```
enum {kSMPSave, kSMPSaveAs, kSMPSaveACopy};

typedef unsigned short SMPSaveType;
```

Constant descriptions

`kSMPSave`	Save an existing file, overwriting the older version, and keeping the file open.
`kSMPSaveAs`	Save the file with a new name, close the original file (if any) without changing it, and open the new file.
`kSMPSaveACopy`	Save a copy of the file with a new name, leaving the original file open.

SPECIAL CONSIDERATIONS

Any time the user changes the content of a letter, you must call the `SMPContentChanged` function immediately. The `SMPBeginSave` function needs this information to operate correctly.

The `SMPBeginSave` function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000B	$1266

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	Disk is full
`fnfErr`	–43	File not found
`fBsyErr`	–47	File is busy
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

Before letting the user change the content of a letter, call the `SMPPrepareToChange` function (page 3-83). If the user changes the content of a letter, you must call the `SMPContentChanged` function (page 3-76) immediately.

When you have finished building your letter, you call the `SMPEndSave` function (described next) to save it. To cancel a save operation any time after you call the `SMPBeginSave` function, call the `SMPEndSave` function with the `okToSave` parameter set to `false`.

Letter file format is briefly discussed in "The Mailer Functions" beginning on page 3-4.

Use the `SMPAddContent` function (page 3-85) to add a standard interchange format block to a letter.

Use the `SMPAddMainEnclosure` function (page 3-90) to add an application document as a main enclosure to a letter.

Use the `SMPImage` function (page 3-88) to add an image block to a letter.

Use the `SMPAddBlock` function (page 3-91) to add a block to a letter.

SMPEndSave

The `SMPEndSave` function saves a letter to disk.

```
pascal OSErr SMPEndSave(WindowPtr window,
                        Boolean okToSave);
```

`window` The window containing the letter to be saved.

`okToSave` A Boolean value that you can use to cancel the process of saving a letter. Specify `false` to cancel the save operation.

DESCRIPTION

After you have used the `SMPBeginSave` function to initiate the process of saving a letter and have added content or a main enclosure to the letter by calling the `SMPAddContent`, `SMPAddBlock`, or `SMPAddMainEnclosure` functions, you call the `SMPEndSave` function to save the letter to disk. You use the `saveType` parameter in the `SMPBeginSave` function to specify which File menu operation this represents: Save, Save As, or Save A Copy. In any case, some version of the file remains open after the file has been saved to disk.

To cancel a save operation any time after you call the `SMPBeginSave` function, call the `SMPEndSave` function with the `okToSave` parameter set to `false`.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1270

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	Disk is full
`kSMPNoMailerInWindow`	–1909	No mailer is in window
`kSMPNoMatchingBegin`	–1913	`SMPBeginSave` was not called

SEE ALSO

You begin the process of saving a letter by calling the `SMPBeginSave` function (page 3-77).

Call the `SMPReadContent` function (page 3-98) to read the standard interchange contents of a letter.

Call the `SMPGetMainEnclosureFSSpec` function (page 3-103) to obtain the file system specification for the main enclosure of a letter.

SMPBeginSend

You must call the `SMPBeginSend` function before you send a letter.

```
pascal OSErr SMPBeginSend(WindowPtr window,
                          OSType creator,
                          OSType fileType,
                          SMPSendOptionsPtr sendOptions,
                          Boolean *mustAddContent);
```

`window` The window containing the letter to be sent.

`creator` The creator for the file, which also becomes the letter's creator for a new letter. Use the same creator that you use for all of your application's documents.

`filetype` The file type, which also becomes the letter type for a new letter. Letters containing only AOCE standard content should be of type `'lttr'`.

`sendOptions`
The pointer to a send-options structure that was returned by the `SMPSendOptionsDialog` function.

`mustAddContent`
A pointer to a Boolean value returned by the function that tells you whether you have to add any blocks or enclosures to the letter. If this parameter is set to `false`, call the `SMPEndSave` function immediately without adding blocks or enclosures to the letter.

DESCRIPTION

When you send a letter, it is in letter file format rather than the document format normally used by your application. A letter consists of a header, data blocks, and enclosures. Every block has a creator and type, and every letter has a creator and type.

Before you send a letter, you should call the `SMPSendOptionsDialog` function to let the user set the send options for that letter. To begin the process of sending a letter, you call the `SMPBeginSend` function. This function prepares a letter file into which you can place your document.

If the user elected to send the letter as a document, you can create a new file format for the document that takes advantage of the block structure of a letter file, or you can save the document in one of your application's native formats to a temporary file on disk and then add that file as the main enclosure to the letter. You should also add a standard interchange format block to the letter. If the user elected to send the letter as an image, you must also add an image block to the letter. The Standard Mail Package does not actually send the letter until you call the `SMPEndSend` function.

If neither your application nor the user has changed the content of a received letter, the `SMPBeginSend` function returns a value of `false` for the `mustAddContent` parameter. In that case you can call the `SMPEndSend` function immediately to send the letter. If you have changed the content of the letter, however, the `mustAddContent` parameter returns `true` and you must build the letter (adding the appropriate combination of blocks, main enclosure, standard interchange format block, and image block) just as if it were a new letter. The Standard Mail Package handles enclosures added by the user. Note that, if you make changes to the enclosed original letter, you invalidate any digital signature.

IMPORTANT

Be sure to call the `SMPContentChanged` function whenever the user changes the content of a letter. If you do not call the `SMPContentChanged` function, then the `SMPBeginSend` function does not know that the letter has been changed and won't return `true` as the value of `mustAddContent`. ▲

SPECIAL CONSIDERATIONS

Any time the user changes the content of a letter, you must call the `SMPContentChanged` function immediately. The `SMPBeginSend` function needs this information to operate correctly.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000A	$1267

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in window
`kMSPCannotSendReceivedLetter`	–1914	Letter is received; cannot send

SEE ALSO

Call the `SMPSendOptionsDialog` function (page 3-73) before calling the `SMPBeginSend` function to let the user set send options.

If the user changes the content of a letter, you must call the `SMPContentChanged` function (page 3-76) immediately.

When you have finished building your letter, you call the `SMPEndSend` function (page 3-84) to send it. To cancel a send operation any time after you call the `SMPBeginSend` function, call the `SMPEndSend` function with the `okToSend` parameter set to `false`.

Letter file format is briefly discussed in "The Mailer Functions" beginning on page 3-4.

Use the `SMPAddContent` function (page 3-85) to add a standard interchange format block to a letter.

Use the `SMPAddMainEnclosure` function page 3-90) to add an application document as a main enclosure to a letter.

Use the `SMPImage` function (page 3-88) to add an image block to a letter.

Use the `SMPAddBlock` function (page 3-91) to add other blocks to a letter.

SMPPrepareToChange

The `SMPPrepareToChange` function checks whether the letter has any digital signatures that might be invalidated if the user changes the content.

```
pascal OSErr SMPPrepareToChange(WindowPtr window)
```

`window` A pointer to the window containing the mailer.

DESCRIPTION

Before making a change in the content of a letter, call the `SMPPrepareToChange` function. If the letter has any digital signatures that might be invalidated by the change, the function displays a dialog box alerting the user and providing a chance to cancel the change. If the user does not cancel the change, you should implement the change and then call the `SMPContentChanged` function. If the user cancels the change, the function returns the `userCanceledErr` result code.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1289

RESULT CODES

`noErr`	0	No error
`userCanceledErr`	–128	User clicked Cancel in dialog box
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

After changing the content of a letter, call the `SMPContentChanged` function (page 3-76).

SMPEndSend

The `SMPEndSend` function sends a letter.

```
pascal OSErr SMPEndSend(WindowPtr window,
                        Boolean okToSend);
```

`window` The window containing the letter to be sent.

`okToSend` A Boolean value that you can use to cancel the process of sending a letter. Specify `false` to cancel the send operation.

DESCRIPTION

After you have used the `SMPBeginSend` function to initiate the process of sending a letter and have created the letter by calling some or all of the `SMPAddContent`, `SMPAddBlock`, `SMPImage`, and `SMPAddMainEnclosure` functions, you call the `SMPEndSend` function to send the letter.

To cancel a send operation any time after you call the `SMPBeginSend` function, call the `SMPEndSend` function with the `okToSend` parameter set to `false`.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1271

RESULT CODES

`noErr`	0	No error
`userCanceledErr`	–128	User clicked Cancel button
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPNoMatchingBegin`	–1913	`SMPBeginSend` was not called

SEE ALSO

You begin the process of sending a letter by calling the `SMPBeginSend` function (page 3-81).

To read the standard interchange contents of a letter you call the `SMPOpenLetter` function (page 3-94) and `SMPReadContent` function (page 3-98).

SMPAddContent

The `SMPAddContent` function adds a segment of a standard interchange format block to a letter.

```
pascal OSErr SMPAddContent(WindowPtr window,
                MailSegmentType segmentType,
                Boolean appendFlag,
                void *buffer,
                unsigned long bufferSize,
                StScrpRec *textScrap,
                Boolean startNewScript,
                ScriptCode script);
```

`window` The window containing the letter.

`segmentType`
A constant that indicates the type of data segment that you want to add to the letter. Letter segments may be text, picture, sound, styled text, or QuickTime movies. You can specify only one segment type in this parameter each time you call the `SMPAddContent` function. It may be any of the following constants:

`kMailTextSegmentType`
Text segment

`kMailPictSegmentType`
Picture segment

kMailSoundSegmentType
Sound segment

kMailStyledTextSegmentType
Styled text segment

kMailMovieSegmentType
Movie segment

The Standard Mail Package also defines another MailSegmentType constant, kMailInvalidSegmentType, which you can use to initialize a variable, for example, in a type-safe manner without indicating that a valid segment has been passed.

appendFlag
A Boolean value that indicates whether you want the SMPAddContent function to write the data in your buffer to a new segment or append it to the current segment. Set this parameter to false when you first call the SMPAddContent function. On subsequent calls to the function, set this parameter to false if you want to start a new segment. Set this parameter to true if you want to append the data in your buffer to the segment currently being written by the SMPAddContent function.

buffer A pointer to the data you want to add to the letter.

bufferSize
The number of bytes of data you want to add to the letter.

textScrap A pointer to an StScrpRec structure that contains style information. You must provide this style information when your buffer contains styled text. Set this parameter to nil if you are not passing styled text data to the function.

startNewScript
A Boolean value that indicates whether the text in your buffer uses a new character set. Set this parameter to true each time you call the SMPAddContent function to start a text segment. After that, set this parameter to true only if the data in your buffer is in a different character set than the data you previously provided to the function for that segment. The function ignores this parameter when you set the segmentType parameter to any value other than kMailTextSegmentType or kMailStyledTextSegmentType.

script A value that indicates the character set (Roman, Arabic, Kanji, etc.) of the data in your buffer. If you set the startNewScript parameter to true, set this parameter to the code for the text segment's character set. You cannot use the values smSystemScript or smCurrentScript for this parameter. The SMPAddContent function ignores this parameter when you set the segmentType parameter to any value other than kMailTextSegmentType or kMailStyledTextSegmentType.

DESCRIPTION

After you have called the SMPBeginSend or SMPBeginSave function, you can call the SMPAddContent function to add standard interchange format data to the letter that is in the window that you specify. The first time you call the function for a given letter, it

creates a new block and puts the data into the block. You then call the function repeatedly until you have finished adding standard interchange format data to the letter. Each time you call the `SMPAddContent` function, it adds data to that same block.

A standard interchange format block consists of data segments, each of a specific type. You add one segment or a portion of a segment of data each time you call the `SMPAddContent` function. The function adds the segments in the order that you provide them. A single letter may contain more than one segment of a given type.

A text segment contains one or more script runs. A script run is a string of text in the same character set. The `SMPAddContent` function can accommodate only one script at a time. Therefore, if you want to create a segment that contains several script runs, you must call this function once for each script run in the segment. Use the `script` parameter to specify the character set of the script run. Set the `startNewScript` parameter to `true` when you start a new text segment and to begin a new script run in the current text segment. To append text to the current script run, set the `startNewScript` parameter to `false` and the `appendFlag` parameter to `true`. If you add a segment of styled text, you must provide the style information in the `textScrap` parameter.

You cannot specify the values `smSystemScript` or `smCurrentScript` for the `script` parameter. To obtain the system script, call the `GetScriptManagerVariable` function with a selector of `smSysScript`. To obtain the current script, call the `FontScript` function.

Because font numbers are local to a given Macintosh computer, the fonts originally used in a letter might be different from those with the same font numbers on the receiving computer. For this reason, the `SMPAddContent` function creates a font table that associates a font name with each font number in the standard interchange format block of the letter. When you receive a letter, you can use the `SMPGetFontNameFromLetter` function to recover the names of the fonts originally used in a letter.

Once you begin creating a letter's standard interchange format block, you must not call other Standard Mail Package functions until you finish writing that block.

The data for picture segments must be in PICT format.

The data for sound segments must be in Audio Interchange File Format (AIFF).

The data for text and styled text segments must consist of 1-byte or 2-byte character codes, depending on the value in the `script` parameter. For styled text you must also provide a pointer to an `StScrpRec` structure in the `textScrap` parameter.

The data for QuickTime movie segments must be in the QuickTime movie format (`'MooV'`).

ASSEMBLY LANGUAGE INFORMATION

Parameter count	Routine selector
$000D	$127A

RESULT CODES

noErr	0	No error
dskFulErr	–34	Disk is full
memFullErr	–108	Not enough room in heap zone
kSMPShouldNotAddContent	–1903	You cannot add content to this letter
kSMPNoMailerInWindow	–1909	No mailer is in specified window

SEE ALSO

See "Summary of the Script Manager" at the end of the "Script Manager" chapter of *Inside Macintosh: Text,* for a list of script code constants.

See *Inside Macintosh: Imaging With QuickDraw* for more information about PICT images.

See *Inside Macintosh: Sound* for more information about AIFF.

The `StScrpRec` structure is described in *Inside Macintosh: Text* in the chapter "TextEdit."

The `StGetScriptManagerVariable` function and `FontScript` function are described in *Inside Macintosh: Text* in the chapter "Script Manager."

The `SMPReadContent` function is described on page 3-98.

You can use the `SMPGetFontNameFromLetter` function (page 3-102) to recover the names of the fonts originally used in a letter.

Call the `SMPAddMainEnclosure` function (page 3-90) to add a main enclosure to a letter. Call the `SMPImage` function (described next) to add an image to the letter. Use the `SMPAddBlock` function (page 3-91) to add other blocks to a letter.

SMPImage

The `SMPImage` function adds an image of a document to a letter.

```
pascal OSErr SMPImage (WindowPtr window,
                       SMPDrawImageProcPtr drawImageProc,
                       long imageRefCon,
                       Boolean supportsColor);
```

`window` The window containing the letter.

`drawImageProc`
A pointer to your image-drawing routine. If you want to send a letter as an image, you must provide a routine to draw the image. The procedure declaration for this routine is described on page 3-123.

`imageRefCon`
A reference constant for your use. The function passes this constant to your image-drawing routine.

`supportsColor`

A Boolean value that indicates whether the procedure pointed to by the `drawImageProc` parameter is capable of drawing in color. The Standard Mail Package provides a color graphics port to your image-drawing routine only if you specify `true` for the `supportsColor` field and the user has color QuickDraw.

DESCRIPTION

You can use the Standard Mail Package to send a letter as an image. You use the `SMPImage` function to create an image from your document and add it to a letter. When you call the `SMPImage` function, you provide a pointer to your drawing routine. The `SMPImage` function calls the drawing routine to draw the image of your document.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1282

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	Disk is full
`memFullErr`	–108	Not enough room in heap zone
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The procedure declaration for your image-drawing routine is on page 3-123.

Mailers are described in "The Mailer Functions" beginning on page 3-4.

Call the `SMPSendOptionsDialog` function (page 3-73) to let the user set send options.

Call the `SMPBeginSend` function (page 3-81) to start the process of sending a letter that contains a mailer.

Call the `SMPAddMainEnclosure` function (described next) to add a main enclosure to a letter. Call the `SMPAddContent` function (page 3-85) to add standard interchange format content to a letter. Use the `SMPAddBlock` function (page 3-91) to add other blocks to a letter.

When you have finished building your letter, you call the `SMPEndSend` function (page 3-84) to send it.

SMPAddMainEnclosure

The `SMPAddMainEnclosure` function adds a main enclosure to a letter.

```
pascal OSErr SMPAddMainEnclosure(WindowPtr window,
                                 const FSSpec *enclosure);
```

`window` The window containing the letter.

`enclosure` A pointer to a file system specification structure that identifies the file that you want to enclose.

DESCRIPTION

When you are creating a letter, you can include a document in one of your application's native formats by first saving the document to disk and then calling the `SMPAddMainEnclosure` function to add the document to the letter as a main enclosure. If you must create a temporary file for this operation, create it in the Temporary Items folder at the root level of the startup volume. The main enclosure is not listed as an enclosure in the mailer.

SPECIAL CONSIDERATIONS

When you save the letter, the file system specification for the main enclosure changes so that the `FSSpec` structure you specified in the `enclosure` parameter is no longer valid. Use the `SMPGetMainEnclosureFSSpec` function to obtain the file system specification of the main enclosure of a letter.

ASSEMBLY LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$127D

RESULT CODES

`noErr`	0	No error
`fnfErr`	–43	File not found
`memFullErr`	–108	Not enough room in heap zone
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

Call the `SMPAddContent` function (page 3-85) to add standard interchange format content to a letter. Call the `SMPImage` function (page 3-88) to add an image to the letter. Use the `SMPAddBlock` function (page 3-91) to add other blocks to a letter.

You can use the `SMPGetMainEnclosureFSSpec` function (page 3-103) to obtain the `FSSpec` structure for the main enclosure of a letter.

The Temporary Items folder is described in the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*.

SMPAddBlock

The `SMPAddBlock` function adds data to a block in a letter.

```
pascal OSErr SMPAddBlock(WindowPtr window,
                         const OCECreatorType *blockType,
                         Boolean append,
                         void *buffer,
                         unsigned long bufferSize,
                         MailBlockMode mode,
                         unsigned long offset);
```

`window` The window containing the letter.

`blockType` A pointer to a data structure that specifies the creator and type of the block you want to add. You may specify any value in the `msgCreator` field of the structure; usually your application signature. The `msgType` field identifies the type of block. You can define your own block types to serve your purposes. Apple Computer, Inc., reserves all block types consisting entirely of lowercase letters.

`append` A Boolean value that indicates whether you want the `SMPAddBlock` function to append the data in your buffer to the current block. Set this parameter to `false` when you call the function to start a new block. If you set this parameter to `true`, the function uses the `mode` and `offset` parameters to determine where to start writing.

`buffer` A pointer to your data buffer.

`bufferSize`
The number of bytes of data to write to the block.

`mode` The mode in which the offset parameter is to be interpreted. The function uses this field to determine whether to begin writing data relative to the end of the last data written, to the beginning of the message, or to the end of the block. See the discussion following these parameter descriptions for details. The function ignores this parameter if you set the `append` parameter to `false`.

`offset` An offset that the function uses when it calculates the starting point of the write operation. Set this value to 0 when you start a new block. See the following discussion for details. The function ignores this parameter if you set the `append` parameter to `false`.

DESCRIPTION

You call the `SMPAddBlock` function to write data into a block whose type you specify in the `blockType` field.

You can write data to a block that is too large to be written all at once by setting the `append` parameter to `true` after the first time you call the function and then calling the function repeatedly until you have written the entire block.

The Standard Mail Package uses a *mark* to point to the current location within a block that you are writing. After the `SMPAddBlock` function completes, the mark points to the end of the last byte written.

You use the `mode` and `offset` parameters to specify the point in the block at which the `SMPAddBlock` function starts writing. You can set the `mode` parameter to any one of the following values:

```
enum {
   kMailFromStart = 1,
   kMailFromLEOB  = 2,
   kMailFromMark  = 3
};
```

Constant descriptions

`kMailFromStart` The function interprets the value in the `offset` parameter as an offset from the beginning of the block. When you use this mode, you cannot set the `offset` parameter to a negative value.

`kMailFromLEOB` The function interprets the value in the `offset` parameter as an offset from the current end of the block. The offset must always be negative and cannot extend beyond the beginning of the block.

`kMailFromMark` The function interprets the value in the `offset` parameter as an offset from the current position of the mark. Use a negative offset value to indicate a starting point prior to the current position of the mark and a positive offset value to indicate a starting point following the current position of the mark. You cannot specify a negative offset that extends beyond the beginning of the block.

To use the `SMPAddBlock` function to write data in several pieces sequentially into a block, call the function as many times as necessary, setting the `mode` parameter to `kIPMFromMark` and the `offset` parameter to 0 each time.

You can overwrite data you have already written to a block but cannot modify a completed block once you start a new block.

SPECIAL CONSIDERATIONS

Once you begin writing a block in a letter, you must finish writing the block, calling the `SMPAddBlock` function as many times as necessary to complete the block before starting another block or calling the `SMPAddContent` or `SMPAddMainEnclosure` functions.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000C	$127F

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The `OCECreatorType` structure is described in "Creator Type Structure" on page 3-28.

Call the `SMPAddMainEnclosure` function (page 3-90) to add a main enclosure to a letter. Call the `SMPAddContent` function (page 3-85) to add standard interchange format content to a letter. Call the `SMPImage` function (page 3-88) to add an image to the letter.

Reading Mail

If you are retrieving mail using the Standard Mail Package, use the `SMPOpenLetter` function (page 3-94) to gain access to an existing letter. If the letter is in the In Tray, you can use the use the `SMPGetLetterInfo` function before you call `SMPOpenLetter`. The `SMPGetLetterInfo` function returns the name and type of the letter. Once you have opened a letter in the In Tray, you can use the `SMPGetNextLetter` function (page 3-97) to open the next or preceding letter in the tray.

While a letter is open, you can examine the standard interchange contents of the letter by calling the `SMPReadContent` function (page 3-98) and can examine the letter's main enclosure by calling the `SMPGetMainEnclosureFSSpec` function (page 3-103). You can use the `SMPGetFontNameFromLetter` function (page 3-102) to determine the original fonts used in the standard interchange content block of a letter. You can use the `SMPEnumerateBlocks` function (page 3-104) to list all the blocks in a letter and the `SMPReadBlock` function (page 3-106) to read any block in a letter, including an image block.

SMPGetLetterInfo

The `SMPGetLetterInfo` function returns information about a letter in the In Tray.

```
pascal OSErr SMPGetLetterInfo(LetterSpec *mailboxSpec,
                              SMPLetterInfo *info);
```

`mailboxSpec`

A pointer to a letter-specification structure. The `LetterSpec` structure is defined on page 3-35.

`info` A pointer to a letter information structure. The `SMPLetterInfo` structure is defined on page 3-27.

DESCRIPTION

The `SMPGetLetterInfo` function lets you determine the creator and letter type of a letter in the In Tray, together with the subject and sender of the letter. You can use this information to title windows or in a dialog box if you cannot open the letter for some reason.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$128A

RESULT CODES

`noErr`	0	No error
`kSMPMailboxNotFound`	–1904	Cannot find mailbox

SEE ALSO

Use the `SMPOpenLetter` function (described next) to open a letter for reading.

SMPOpenLetter

The `SMPOpenLetter` function opens a letter so you can read the contents.

```
pascal OSErr SMPOpenLetter(const LetterDescriptor *letter,
                           WindowPtr window,
                           Point upperLeft,
                           Boolean canContract,
                           Boolean initiallyExpanded,
                           const PrepareMailerForDrawingProcPtr
                                          prepareMailerForDrawingCB,
                           long clientData);
```

`letter` A pointer to the letter descriptor of the letter you want to open. The letter descriptor specifies whether the letter is on disk or in the In Tray and provides the file system specification structure or letter-specification structure of the letter.

`window` The window in which you want to display the opened letter. This window must not contain a mailer at the time you call the `SMPOpenLetter` function.

`upperLeft` The upper-left corner of the mailer in your window's local coordinates. This position is normally (0, 0).

`canContract`
A Boolean value that specifies whether it should be possible to contract and expand the mailer. Specify `true` if you want the mailer to have this ability.

`initiallyExpanded`
A Boolean value that indicates whether you want the mailer displayed initially in its expanded or contracted state. Specify `true` to display the mailer initially expanded.

`prepareMailerForDrawingCB`
A pointer to your drawing-preparation function. If you change the clip region, coordinates, or other aspects of your window's graphics port, you must provide this function to restore the graphics port to a standard state so that the Standard Mail Package can draw a mailer in your window. The drawing-preparation function is described on page 3-122. Specify `nil` for this parameter if you are not providing a drawing-preparation function.

`clientData`
Reserved for your use. The `SMPOpenLetter` function passes this value unaltered to your callback routine.

DESCRIPTION

You call the `SMPOpenLetter` function when you receive an Apple event to open a letter or when the user chooses the Open command. This function displays a mailer in the window you specify and opens the letter so you can read its contents.

The user can double-click an icon in the In Tray to open a letter or double-click the icon for a letter file on disk. Your Open item in the File menu should also open letter files on disk. There is no way to open a letter in the In Tray from a menu in your application; the user must use the Finder to open the letter. The Finder determines the owner of the letter and sends an `'aevt' 'odoc'` Apple event to the appropriate application, which can then open it. The Apple event contains the letter-specification structure, which you put in the letter descriptor and pass to the `SMPOpenLetter` function. If the user chooses the Open command to open the letter, you can get the file system specification structure from the Standard File Package.

Whether a letter is on disk or not, the Standard Mail Package treats the letter's enclosures as if they are stored in a folder in an external file system. That folder contains all of the enclosures added by the user through the mailer or by the `SMPAddAttachment` function, and might contain the letter's main enclosure, if any.

(You can use the `SMPAddMainEnclosure` function to add a main enclosure to a letter. The main enclosure is not displayed in the Enclosures field of a mailer.)

The Standard Mail Package displays the enclosures added by the user in the Enclosures field of the mailer and handles the user interface for those enclosures. To obtain the file system specification structure of the main enclosure to a letter, use the `SMPGetMainEnclosureFSSpec` function. Use the `SMPGetListItemInfo` function to get the file system specification for the enclosures added by the user.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000C	$1268

RESULT CODES

`noErr`	0	No error
`kSMPMailerAlreadyInWindow`	–1911	Specified window has a mailer
`kMailInvalidSeqNum`	–15041	Invalid letter sequence number

SEE ALSO

The `LetterDescriptor` data type is described in "Letter Descriptor" on page 3-27.

Use the `SMPGetMainEnclosureFSSpec` function (page 3-103) to obtain the file system specification structure of the main enclosure of a letter.

Use the `SMPGetListItemInfo` function (page 3-113) to get the file system specification for the enclosures added by the user.

You can use the `SMGetComponentInfo` function (page 3-111) to obtain the file system specification structure of the enclosures folder for the letter.

You can use the `SMPGetNextLetter` function (described next) to determine which letter in the In Tray is the oldest unread letter.

SMPGetNextLetter

The `SMPGetNextLetter` function returns the letter descriptor of the In Tray item to be opened next.

```
pascal OSErr SMPGetNextLetter(
                            OSType *typesList,
                            short numTypes,
                            LetterDescriptor *adjacentLetter);
```

`typesList` A pointer to a list of letter types and file types. The function returns letter descriptors only for items with the letter types and file types specified in this list. Letters containing only AOCE standard content are of type `'lttr'`. Use the wildcard letter type `'ltr*'` if you want the function to return letter descriptors for all the items in the In Tray.

`numTypes` The number of letter types and file types in the types list pointed to by the `typesList` parameter.

`adjacentLetter`
The letter descriptor returned by the function identifying the next item to open.

DESCRIPTION

The letter descriptor returned by the `SMPGetNextLetter` function is that of the oldest unread letter in the In Tray. You can use this function to open letters in age sequence.

You can use the `typesList` parameter to specify what letter types and file types the function should include when it returns a letter descriptor.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0008	$1286

RESULT CODES

`noErr`	0	No error
`kSMPMailboxNotFound`	–1904	Cannot find mailbox
`kSMPNoNextLetter`	–1905	There is no next letter in the In Tray

SEE ALSO

Use the `SMPOpenLetter` function (page 3-94) to open a letter for reading.

The letter descriptor (`LetterDescriptor` data type) is described in "Letter Descriptor" on page 3-27.

SMPReadContent

The `SMPReadContent` function reads a segment from a letter's standard interchange format block.

```
pascal OSErr SMPReadContent(WindowPtr window,
               MailSegmentMask segmentTypeMask,
               void *buffer,
               unsigned long bufferSize,
               unsigned long *dataSize,
               StScrpRec *textScrap,
               ScriptCode *script,
               MailSegmentType *segmentType,
               Boolean *endOfScript,
               Boolean *endOfSegment,
               Boolean *endOfContent,
               long *segmentLength,
               long *segmentID);
```

`window` The window containing the letter.

`segmentTypeMask`
The type of segment that you want to retrieve. You can request a combination of segment types by performing a bitwise OR operation on the following constants:

`kMailTextSegmentMask`
Text segment

`kMailPictSegmentMask`
Picture segment

`kMailSoundSegmentMask`
Sound segment

`kMailStyledTextSegmentMask`
Styled text segment

`kMailMovieSegmentMask`
Movie segment

You can request any combination of segment types, except that you cannot combine the `kMailTextSegmentMask` and `kMailStyledTextSegmentMask` constants in the same request. If you request styled text segments, the function returns both plain text and

styled text segments. If you request plain text segments, it returns any plain text segments that are in the letter and also converts styled text segments to plain text segments and returns them to you.

The `SMPReadContent` function reads this parameter only the first time you call it for a given letter. The next and subsequent times you call this function for the same letter, it ignores this parameter. The function returns data of a single segment type each time you call it.

`buffer` A pointer to the buffer you are providing to hold the data read from the letter.

`bufferSize`
The size of the buffer you are providing.

`dataSize` A pointer to the amount of data returned in your buffer.

`textScrap` A pointer to an `StScrpRec` structure. You must allocate this pointer. Set the first field of this structure (`scrpNStyles`) to the number of styles your buffer can hold. When the function writes styled text to your buffer, it returns style information in this structure and sets the `scrpNStyles` field to the actual number of styles returned.

`script` A pointer to the script code, which indicates the character set (Roman, Arabic, Kanji, etc.) of the text that the function placed in your buffer. If the function placed nontext data in your buffer, it does not set this parameter and you should ignore it.

`segmentType`
A constant that indicates the type of data segment that the `SMPReadContent` function returned in your buffer. It may be any of the following constants:

`kMailTextSegmentType`
Text segment

`kMailPictSegmentType`
Picture segment

`kMailSoundSegmentType`
Sound segment

`kMailStyledTextSegmentType`
Styled text segment

`kMailMovieSegmentType`
Movie segment

The Standard Mail Package also defines another `MailSegmentType` constant, `kMailInvalidSegmentType`, which you can use to initialize a variable, for example, in a type-safe manner without indicating that a valid segment has been passed.

`endOfScript`
A pointer to a Boolean value returned by the function that indicates whether the text placed in your buffer is the end of a script run. A script run is a sequence of text in a single character set. If there is more text in the current script run, the function sets this parameter to `false`.

`endOfSegment`
: A pointer to a Boolean value returned by the function that indicates whether you have received all of the data in the segment. The `SMPReadContent` function sets this parameter to `true` when it has returned all of the data in a given segment. It sets this parameter to `false` when there is more data in that segment to return.

`endOfContent`
: A pointer to a Boolean value returned by the function that indicates if there is more data in the standard interchange format block of the letter to be read. If the `SMPReadContent` function has returned the entire contents of the block, it sets the `endOfContent` parameter to `true`; otherwise, it sets this parameter to `false`.

`segmentLength`
: A pointer to the size of the current segment. The `SMPReadContent` function returns a valid value for this parameter the first time you call the function to read a particular segment.

`segmentID`
: A pointer to the ID of this segment. If you specify 0 for this parameter, the `SMPReadContent` function reads the next segment sequentially, returning the segment ID in this parameter. If you specify a value, `SMPReadContent` reads the segment with the specified ID. Note that this number is not an index number; it is an ID that is unique for the beginning of each segment. The number returned by the function in this parameter when you continue reading in the middle of a segment is undefined. You must set this parameter to 0 the first time you call the function.

DESCRIPTION

You call the `SMPReadContent` function to read some or all of a letter's standard interchange format block. You must call the `SMPOpenLetter` function before the first time you call the `SMPReadContent` function for a given letter. You then call the `SMPReadContent` function repeatedly to read all of the segments of the types you specified the first time you called the function. Once the `SMPReadContent` function has returned `true` for the `endOfContent` parameter, you must call the `SMPOpenLetter` function again before you can call the `SMPReadContent` function again.

The `SMPReadContent` function examines the value of the `segmentTypeMask` parameter the first time you call it for a given letter and uses that same value until you start the sequence over by calling the `SMPOpenLetter` function again. The `SMPReadContent` function returns segments in the order that they are stored in the letter.

If you request styled text segments, the function returns both plain text and styled text segments. If you request plain text segments, it returns any plain text segments that are in the letter and also converts styled text segments to plain text segments and returns them to you.

A text segment contains one or more script runs. A script run is a string of text in the same character set. When the `SMPReadContent` function returns text data (that is, when the function sets the `segmentType` parameter to `kMailTextSegmentType`), it

indicates the character set by setting the `script` parameter. The function identifies the end of a script run by setting the `endOfScript` parameter to `true`.

The `SMPReadContent` function returns, in the `dataSize` parameter, a pointer to the actual number of bytes written to your buffer. If your buffer is not large enough to hold all of the data in a segment, the function sets the `endOfSegment` parameter to `false`. You can call the function again to continue reading data from that segment.

If a single segment of styled text contains more styles than your `StScrpRec` structure can hold, the `SMPReadContent` function stops writing data to your buffer and sets the `endOfSegment` parameter to `false`. You can use the `dataSize` parameter to determine how many bytes of text were written to your buffer. The next time to call the function, it continues writing text from the same segment into your buffer and putting text styles in your `StScrpRec` structure. In this case, the offsets in the `scrpStartChar` field of the script table of the `StScrpRec` structure apply only to the data currently in your data buffer, not to the offsets in the original segment in the letter.

For example, suppose that the next segment in the letter to be read is a styled text segment that is 120 bytes long and contains 12 different styles. The 11th style starts at an offset of 90 (that is, at the 91st byte of the segment). Suppose further that your text buffer is 200 bytes but your `StScrpRec` structure can hold only 10 styles. In this case, the `SMPReadContent` function stops writing data to your buffer after it has placed 10 styles in your `StScrpRec` structure. Because these 10 styles applied to the first 90 bytes of text, the `dataSize` parameter indicates that 90 bytes of data were written to your buffer and the `endOfSegment` parameter is `false`.

The next time you call the `SMPReadContent` function, it writes the last 30 bytes of text into your buffer and puts the last two styles into your `StScrpRec` structure. It returns a value of 2 in the `scrpNStyles` field of your `StScrpRec` structure and sets the `endOfSegment` parameter to `true`. In this case, the first offset in the `scrpStartChar` field of the script table of the `StScrpRec` structure is 0, indicating that the first style in the text scrap starts with the first byte of text currently in your buffer. (The offset is *not* 90, as it would have been for this portion of text had your `StScrpRec` structure been able to hold all of the styles at once.)

The data for picture segments is in PICT format.

The data for sound segments is in Audio Interchange File Format (AIFF).

The data for text and styled text segment consists of 1-byte or 2-byte character codes, depending on the value in the `script` parameter. For styled text the function also returns a pointer to an `StScrpRec` structure in the `textScrap` parameter.

The data for QuickTime movie segments must be in the QuickTime movie format (`'MooV'`).

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0019	$127B

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kMailMalformedContent`	–15061	A mailed structure is malformed

SEE ALSO

See *Inside Macintosh: Imaging With QuickDraw* for more information about PICT images.

See *Inside Macintosh: Sound* for more information about AIFF.

The `StScrpRec` structure is described in the chapter "TextEdit" of *Inside Macintosh: Text*.

SMPGetFontNameFromLetter

The `SMPGetFontNameFromLetter` function converts the font numbers in the standard interchange format block of a letter into font names.

```
pascal OSErr SMPGetFontNameFromLetter(WindowPtr window,
                                      short fontNum,
                                      str255 fontName,
                                      Boolean doneWithFontTable);
```

`window` The window containing the letter.

`fontNum` The font number you read from the text scrap (the `TextEdit` structure) for the text.

`fontName` The name of the font associated with the font number.

`doneWithFontTable`
A Boolean value that you set to indicate that this is the last request for a font name.

DESCRIPTION

You can use the `SMPGetFontNameFromLetter` function to recover the names of the fonts originally used in a letter. Because font numbers are local to a given Macintosh computer, the fonts originally used in a received letter might be different from those with the same font numbers on the local computer. For this reason, when the Standard Mail Package sends a letter, it creates a font table that associates a font name with each font number in the standard interchange format block of the letter.

To recover the font names, you must first call the `SMPReadContent` function to read the standard content block of the letter, and then read the font numbers from the `StScrpRec` structure associated with each styled-text segment in that block. Then you can call the `SMPGetFontNameFromLetter` function once for each font number.

Set the `doneWithFontTable` parameter to `true` the last time you call the `SMPGetFontNameFromLetter` function. Doing so signals the Standard Mail Package to release the memory it has reserved for the font table.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$127C

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The `StScrpRec` structure is described in the chapter "TextEdit" of *Inside Macintosh: Text*.

SMPGetMainEnclosureFSSpec

The `SMPGetMainEnclosureFSSpec` function returns the file specification of the main enclosure file for a letter.

```
pascal OSErr SMPGetMainEnclosureFSSpec (WindowPtr window,
                                        FSSpec *enclosureDir);
```

`window` The window containing the letter.

`enclosureDir`
A pointer to the file system specification structure of the main enclosure.

DESCRIPTION

You can call the `SMPGetMainEnclosureFSSpec` function to get the file system specification for the main enclosure file for a letter. The main enclosure contains the letter's content, usually in your application's native document format. You can then use standard File Manager routines to open and read the main enclosure. The file system specification returned by this function is valid until the mailer is disposed of or until the next time the user saves the letter. You must call the `SMPOpenLetter` function before you call the `SMPGetMainEnclosureFSSpec` function.

If the letter does not contain a main enclosure, the function returns the result code `fnfErr` (file not found).

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$127E

RESULT CODES

`noErr`	0	No error
`fnfErr`	–43	File not found
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You use the `SMPAddMainEnclosure` function (page 3-90) to add a main enclosure to a letter.

File Manager routines are described in *Inside Macintosh: Files*.

SMPEnumerateBlocks

The `SMPEnumerateBlocks` function returns information about the blocks in a letter.

```
pascal OSErr SMPEnumerateBlocks (WindowPtr window,
                                 unsigned short startIndex,
                                 void *buffer,
                                 unsigned long bufferSize,
                                 unsigned long *dataSize,
                                 unsigned short *nextIndex,
                                 Boolean *more);
```

`window` The window containing the letter.

`startIndex`
The sequence number of the next block for which you want the function to return information. Sequence numbers start with 1. When you call the `SMPEnumerateBlocks` function and there is insufficient space in the buffer you provide to hold information about all of the remaining blocks, the function returns, in the `nextIndex` parameter, the sequence number of the next block. Use that number as the value of the `startIndex` parameter the next time you call the function.

`buffer` A pointer to a buffer you provide to hold the information returned by the function. The block information is in the form of a count byte, indicating the number of blocks in the letter, followed by a block information structure for each block.

`bufferSize`
The length, in bytes, of the buffer you are providing.

`dataSize` The address at which the function places the number of bytes written to your buffer.

`nextIndex` The address at which the function places the sequence number of the first block whose information did not fit into your buffer. The function sets this field when your buffer is too small to hold all the information you requested. If there is no more information to return, the function sets the sequence number to 0.

`more` A Boolean value returned by the function indicating whether there is more block information to be returned. If your buffer is too small to hold all of the information that you requested, the `SMPEnumerateBlocks` function sets this parameter to `true` and returns, in the `nextIndex` parameter, the sequence number of the next item to be returned.

DESCRIPTION

You can use the `SMPEnumerateBlocks` function to determine the number of blocks that are contained in a letter and each block's type and size. You can use this information to read specific blocks in the letter. You must call the `SMPOpenLetter` function before the first time you call the `SMPEnumerateBlocks` function for a given letter.

Apple Computer, Inc., reserves all block types that consist of all lowercase letters for its own use. Use the `SMPReadBlock` function to read image blocks and blocks of types that you define. Use the `SMPReadContent` function to read the standard interchange format block.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000D	$1281

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

Use the `SMPReadBlock` function (described next) to read the contents of a block.

Use the `SMPReadContent` function (page 3-98) to read the standard interchange format block in a letter.

SMPReadBlock

The `SMPReadBlock` function reads a block from a letter that you specify.

```
pascal OSErr SMPReadBlock (WindowPtr window,
                           const OCECreatorType *blockType,
                           unsigned short blockIndex,
                           void *buffer,
                           unsigned long bufferSize,
                           unsigned long dataOffset,
                           unsigned long *dataSize,
                           Boolean *endOfBlock,
                           unsigned long *remaining);
```

`window` The window containing the letter.

`blockType` A pointer to a structure that specifies the creator and the type of the block that you want to read.

`blockIndex`
The relative position of the block of type `blockType` that you want to read. To read all blocks of a specific block type, set this field to 1 the first time you call the `SMPReadBlock` function and increment it by 1 each subsequent time you call the function until you have read all blocks of that type in the letter.

`buffer` A pointer to your data buffer. The `SMPReadBlock` function writes the information that you request into your buffer and sets the `dataSize` field to the number of bytes written.

`bufferSize`
The length, in bytes, of the buffer you are providing.

`dataOffset`
The offset relative to the beginning of the block of the byte at which you want the `SMPReadBlock` function to begin reading. Set this field to 0 to read from the beginning of the block.

`dataSize` A pointer to the number of bytes written to your buffer.

`endOfBlock`
A pointer to a Boolean value that indicates if the `SMPReadBlock` function has reached the end of the block. If the buffer that you provide is not large enough to contain the data remaining in the block, the `SMPReadBlock` function sets this parameter to `false`. You can call the function again with an updated value in the `dataOffset` parameter to retrieve additional data.

`remaining` A pointer to the number of bytes of data remaining in the block. You can use the value returned by this parameter to adjust the size of your data buffer before the next time you call the function. When the function sets the `endOfBlock` parameter to `true`, it sets the number of bytes remaining to 0.

DESCRIPTION

You call the `SMPReadBlock` function to read data from a specific block in a letter. You identify the block that you want to read by the values of the `blockType` and `blockIndex` parameters.

You can use this function to read an image block (a block with creator type `'apml'` and block type `'imag'`) or any block of a type you define.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0012	$1280

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You can use the `SMPEnumerateBlocks` function (page 3-104) to list the block types and sizes of blocks in a letter.

The `OCECreatorType` data structure is described in "Creator Type Structure" on page 3-28.

The section "Image Block Information Structure" on page 3-28 describes how to read an image block.

Printing Mailers

If you are printing or imaging a letter, you should print or image the mailers as cover pages. You use the `SMPPrepareCoverPages` function (described next) to determine the total number of cover pages and the `SMPDrawNthCoverPage` function (page 3-108) to draw each cover page.

SMPPrepareCoverPages

The `SMPPrepareCoverPages` function prepares cover pages for a letter and returns the number of cover pages that are needed to print all of the mailers for the letter.

```
pascal OSErr SMPPrepareCoverPages(windowPtr window,
                                  short *pageCount);
```

`window`	The window for which you want the number of cover pages.
`pageCount`	A pointer to the number of cover pages necessary to print all the mailers in the specified window.

DESCRIPTION

When you print or image a letter, you can print or image the mailers as cover pages. You must call the `SMPPrepareCoverPages` function from within your printing or imaging routine to prepare the cover pages and to determine the number of cover pages before calling the `SMPDrawNthCoverPage` function.

SPECIAL CONSIDERATIONS

The `SMPPrepareCoverPages` function makes a number of calculations that are used by the `SMPDrawNthCoverPage` function. You must make sure that the mailer does not change between the time you call these two functions.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1264

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You call the `SMPDrawNthCoverPage` function, described next, to draw a cover page to the current graphics port.

SMPDrawNthCoverPage

The `SMPDrawNthCoverPage` function draws a cover page for a letter.

```
pascal OSErr SMPDrawNthCoverPage(WindowPtr window,
                                 short pageNumber,
                                 Boolean doneDrawingCoverPages);
```

`window` A pointer to the window for which you want to draw a cover page.

`pageNumber`
The number of the cover page you want to print.

`doneDrawingCoverPages`
A Boolean value that you set to `true` when you call the function for your last cover page. Doing so allows the Standard Mail Package to release the memory that it uses for drawing cover pages.

DESCRIPTION

Before you print or image a letter, you should include the mailers for that letter as cover pages. The `SMPDrawNthCoverPage` function draws or images one cover page. You must use the `SMPPrepareCoverPages` function first to prepare the cover pages and to determine the total number of cover pages for a given letter. You call these functions from within your drawing or imaging routine, and they draw to whatever graphics port you provide. You can use these routines for printing, preparing an image of your letter to be sent as electronic mail, or for display on the screen.

SPECIAL CONSIDERATIONS

The `SMPDrawNthCoverPage` function uses a number of calculations that are made by the `SMPPrepareCoverPages` function. You must make sure that the mailer does not change between the time you call these two functions.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1265

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The `SMPPrepareCoverPages` function is described on page 3-107.

For the sequence of routines you must call to image a letter, see the description of the image-drawing callback routine on page 3-123.

Getting and Setting Information in the Mailer

You can use the functions in this section to determine the contents of the fields of a mailer and to change the information in those fields without user interaction.

The `SMPGetListItemInfo` function (page 3-113) returns information from the Recipients or Enclosures fields of any mailer. The `SMPGetComponentInfo` function (page 3-111) returns information from any other field. Before calling either of these functions, you call the `SMPGetComponentSize` function (described next) to determine the size of the buffer to allocate.

You can put information into the fields of a mailer with the functions `SMPSetSubject` (page 3-116), `SMPSetFromIdentity` (page 3-117), `SMPAddAddress` (page 3-118), and `SMPAddAttachment` (page 3-119).

SMPGetComponentSize

The `SMPGetComponentSize` function returns the size of the buffer that would be required to hold all of the information in a specific field of the mailer you specify.

```
pascal OSErr SMPGetComponentSize(WindowPtr window,
                                 unsigned short whichMailer,
                                 SMPMailerComponent whichField,
                                 unsigned short *size);
```

`window` The window containing the mailer from which you want information.

`whichMailer`
The sequence number of the mailer from which you want information. The original mailer for the letter is number 1, and each forwarding mailer is numbered sequentially.

`whichField`
The field from which you want to extract the information.

`size` A pointer to the number of bytes of data in the field you specified. For all fields except the Recipients and Enclosures fields, you should allocate a buffer of this size and call the `SMPGetComponentInfo` function to obtain the contents of that field. The Recipients and Enclosures fields might contain more data than it is practical to retrieve all at once; see the description of the `SMPGetListItemInfo` function (page 3-113) for more information.

DESCRIPTION

The `SMPGetComponentSize` function returns the number of bytes of data in any of the fields in a mailer. You specify which field by using one of the following constants for the `whichField` parameter: `kSMPFrom`, `kSMPTo`, `kSMPRegarding`, `kSMPSendDateTime`, or `kSMPAttachments`.

If you specify any other value for the `whichField` parameter, the function returns the `kSMPIllegalComponent` result code.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0007	$1277

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in user parameter list
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPIllegalComponent`	–1918	Illegal value for `whichField` parameter

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to get the total number of mailers for a given letter.

Use the `SMPGetComponentInfo` function (described next) to get data from any field except the Recipients and Enclosures fields. Use the `SMPGetListItemInfo` function (page 3-113) to get data from the Recipients and Enclosures fields.

All possible values for the `SMPMailerComponent` data type are shown on page 3-32.

SMPGetComponentInfo

The `SMPGetComponentInfo` function returns information from the From, Subject, and Sent fields of a mailer.

```
pascal OSErr SMPGetComponentInfo(WindowPtr window,
                                 unsigned short whichMailer,
                                 SMPMailerComponent whichField,
                                 void *buffer);
```

`window` The window containing the mailer from which you want information.

`whichMailer`
The sequence number of the mailer from which you want information. The original mailer for the letter is number 1, and each forwarding mailer is numbered sequentially.

`whichField`
The field from which you want to extract the information.

`buffer` A pointer to a buffer you provide into which the function places the information you requested. Use the `SMPGetComponentSize` function to determine what size to make this buffer.

DESCRIPTION

The `SMPGetComponentInfo` function returns information from a mailer. You specify which field by using one of the following constants for the `whichField` parameter: `kSMPFrom` (for the From field), `kSMPRegarding` (for the Subject field), or `kSMPSendDateTime` (for the Sent field).

If you specify any other value for the `whichField` parameter, the function returns the `kSMPIllegalComponent` result code.

If you request information from the Subject field, then the function returns an `RString` structure containing the text of the field.

If you request information from the From field of a draft mailer, the function returns an `AuthIdentity` structure identifying the sender, followed by an `RString` structure containing the text in the From field. If you request information from the From field of a received mailer, the function returns an `OCEPackedRecipient` structure containing the address of the sender. Only the top mailer in a mailer set can be a draft mailer; use the `hasBeenReceived` field of the `SMPMailerState` structure to determine whether the top mailer has been received.

If you request information from the Date field of the mailer, then the function returns a `MailTime` structure. The time is defined with respect to the local computer that records it. The `offset` field in the `MailTime` structure corrects UTC time (also known as Greenwich Mean Time) for the local time zone. The `offset` field is in seconds; it is positive if east of Greenwich and negative if west of Greenwich.

```
typedef struct MailTime {
   UTCTime     time;                /* current UTC (GMT) time */
   UTCOffset   offset;              /* in seconds from GMT */
};

typedef struct MailTime MailTime;

typedef  unsigned long  UTCTime;    /* seconds since 1/1/1904 */
typedef  long           UTCOffset;  /* correct for local time */
```

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0007	$1278

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in user parameter list
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPIllegalComponent`	–1918	Illegal value for `whichField` parameter

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to get the total number of mailers for a given letter and to determine if the top mailer has been received.

Use the `SMPGetComponentSize` function (page 3-110) to determine the size of the buffer to provide. Use the `SMPGetListItemInfo` function (described next) to get data from the Recipients and Enclosures fields.

All possible values for the `SMPMailerComponent` data type are shown on page 3-32.

SMPGetListItemInfo

The `SMPGetListItemInfo` function returns information from the Recipients or Enclosures fields of a mailer.

```
pascal OSErr SMPGetListItemInfo(WindowPtr window,
                                unsigned short whichMailer,
                                SMPMailerComponent whichField,
                                void *buffer,
                                unsigned long bufferLength,
                                unsigned short startItem,
                                unsigned short *itemCount,
                                unsigned short *nextItem,
                                Boolean *more);
```

`window` The window containing the mailer from which you want information.

`whichMailer`
The sequence number of the mailer from which you want information. The original mailer for the letter is number 1, and each forwarding mailer is numbered sequentially.

`whichField`
The field from which you want information; either `kSMPAttachments` or `kSMPTo`.

`buffer` A pointer to a buffer you provide to hold the information returned by the function.

`bufferLength`
The length, in bytes, of the buffer you are providing.

`startItem` The sequence number of the first address or enclosure that the function should return. Sequence numbers start with 0. When you call the `SMPGetListItemInfo` function and there is insufficient space in the buffer you provide to hold all of the remaining items, the function returns, in the `nextItem` parameter, the sequence number of the next item. Use that number for the `startItem` parameter the next time you call the function. If there is insufficient space in the buffer to hold even one item, the number the function returns in the `nextItem` parameter is the same as the number you put in the `startItem` parameter. In that case, you must increase the buffer size before calling the function again.

`itemCount` A pointer to the number of items that the function has placed in the buffer. If the buffer is too small to hold the item you specify in the `startItem` parameter, then the `itemCount` parameter returns 0, and the `more` parameter returns `true`. If you specify `nil` for the `buffer` parameter and 0 for the `bufferLength` parameter, the `itemCount` parameter returns a pointer to the total number of items in the mailer field.

`nextItem` A pointer to the sequence number of the next item to be returned. If the `more` parameter returns `true`, set the `startItem` parameter to the number returned in the `nextItem` parameter and call the function again.

`more` A pointer to a Boolean value returned by the function indicating whether there is more information to be returned. If your buffer was not large enough to hold all of the requested data, the function sets this parameter to `true` and returns, in the `nextItem` parameter, the sequence number of the next item to be returned.

DESCRIPTION

Before you call the `SMPGetListItemInfo` function, call the `SMPGetComponentSize` function with a value of `kSMPTo` or `kSMPAttachments` for the `whichField` parameter. The `SMPGetComponentSize` function returns the total number of bytes of storage space required to hold all of the information you requested. You can then allocate a buffer to hold the data returned by the `SMPGetListItemInfo` function. If you can't (or don't want to) provide a buffer large enough to hold all of the information at once, you can allocate a smaller buffer.

If you cannot allocate a buffer large enough to hold all of the items at once, the function returns the sequence number of the next item in the `nextItem` parameter and it returns `true` for the `more` parameter. If the buffer is not large enough to hold even one item, the sequence number returned in the `nextItem` parameter is the same as the number you passed in the `startItem` parameter. You can then increase the size of your buffer if necessary, set the `startItem` parameter to the number just returned in the `nextItem` parameter, and call the function again.

You can specify either `kSMPTo` or `kSMPAttachments` for the `whichField` parameter.

If you request information from the Recipients field, for each address the function returns a short value indicating the type of address followed by an `OCEPackedRecipient` structure containing the address. The address type can be any of the following values:

```
enum {
   kSMPToAddress =      kMailToBit,
   kSMPCCAddress =      kMailCcBit,
   kSMPBCCAddress =     kMailBccBit
};

typedef MailAttributeID SMPAddressType;
```

If you request information from the Enclosures field, the function returns a file system specification structure (`FSSpec` data type) identifying the letter's enclosure folder. You can then use File Manager routines to determine the contents of that folder.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0010	$1279

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in user parameter list
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPIllegalComponent`	–1918	Illegal value for `whichField` parameter

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to get the total number of mailers for a given letter.

Use the `SMPGetComponentSize` function (page 3-110) to determine the size of the buffer to provide.

Use the `SMPGetComponentInfo` function (page 3-111) to get data from mailer fields other than the Recipients and Enclosures fields.

SMPSetSubject

The `SMPSetSubject` function specifies the subject string for the top mailer in the window you specify.

```
pascal OSErr SMPSetSubject(WindowPtr window,
                           const RString *text);
```

`window` The window containing the mailer.

`text` A pointer to the subject string you want to place in the mailer.

DESCRIPTION

The Standard Mail Package provides a user interface that lets a user enter a subject string in a mailer. You can use the `SMPSetSubject` function to set the subject string directly from your application. You can use this function, for example, to place an initial, default subject string in the subject field of a new mailer.

The `SMPSetSubject` function sets only the string in the most recent mailer for the window you specify, and then only if it is a draft mailer (that is, if it is not a received mailer). You can use the `hasBeenReceived` field of the `SMPMailerState` structure (a parameter of the `SMPGetMailerState` function) to determine whether the mailer is a draft mailer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$126B

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in user parameter list
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPMailerUneditable`	–1912	Mailer cannot be edited
`kSMPSubjectTooBig`	–1925	Subject string exceeds 127 characters

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to determine if the top mailer is for a received letter.

SMPSetFromIdentity

The `SMPSetFromIdentity` function sets the authentication identity for the sender of a letter.

```
pascal OSErr SMPSetFromIdentity(WindowPtr window,
                                AuthIdentity from);
```

`window` The window containing the mailer.

`from` The authentication identity you want to use for that mailer. Specify 0 to use the identity of the most recently authenticated user.

DESCRIPTION

The `SMPSetFromIdentity` function lets you change the contents of the From field of a mailer from within your application. The `SMPSetFromIdentity` function modifies only the most recent mailer in the specified window, and then only if it is not a received mailer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$126C

RESULT CODES

`noErr`	0	No error
`kOCEUnknownID`	–1567	Identity passed is not valid
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPMailerUneditable`	–1912	Mailer cannot be edited

SEE ALSO

Use the `SDPPromptForID` function in the chapter "Standard Catalog Package" in this book to obtain an authentication identity.

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to determine if the top mailer is for a received letter.

SMPAddAddress

The `SMPAddAddress` function adds an address to the Recipients field of a mailer.

```
pascal OSErr SMPAddAddress(WindowPtr window,
                SMPAddressType addrType,
                OCEPackedRecipient *address);
```

`window` The window containing the mailer.

`addrType` The type of address you want to add. You can specify the value `kSMPToAddress` to add a primary addressee, or `kSMPCCAddress` to add a "copy to" addressee.

`address` The address that you want to add to the mailer.

DESCRIPTION

The Standard Mail Package provides a user interface that lets a user enter an address in the Recipients field of a mailer. You can use the `SMPAddAddress` function to add an address directly from your application. You can use this function, for example, to place an initial, default address in the Recipients field of a reply mailer. If you specify an address type of `kSMPCCAddress`, the mailer flags the address as a "copy to" address (see Figure 3-3 on page 3-5). The values of the `SMPAddressType` data type are defined as follows:

```
enum {
   kSMPToAddress =      kMailToBit,
   kSMPCCAddress =      kMailCcBit,
   kSMPBCCAddress =     kMailBccBit
};

typedef MailAttributeID SMPAddressType;
```

The `SMPAddAddress` function adds addresses only to the most recent mailer in the specified window, and then only if it is not a received mailer. You can use the `hasBeenReceived` field of the `SMPMailerState` structure (a parameter of the `SMPGetMailerState` function) to determine whether the top mailer has been received.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0005	$126D

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPMailerUneditable`	–1912	Mailer cannot be edited
`kSMPAddressAlreadyInList`	–1922	Specified address is in Recipients field

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to determine if the top mailer is for a received letter.

You can use the `SMPGetListItemInfo` function (page 3-113) to get the addresses that the user entered in the Recipients field of a mailer.

SMPAddAttachment

The `SMPAddAttachment` function adds a disk file as an enclosure to a letter.

```
pascal OSErr SMPAddAttachment(WindowPtr window,
                              const FSSpec *attachment);
```

`window` The window containing the mailer.

`attachment`
A pointer to the file system specification structure of the disk file that you want to add as an enclosure.

DESCRIPTION

The Standard Mail Package provides a user interface that lets a user add a disk file or folder as an enclosure to a letter. You can use the `SMPAddAttachment` function to add an enclosure directly from your application in case you want to provide an Add Enclosures command. The `SMPAddAttachment` function adds enclosures only to the most recent mailer in the specified window, and then only if it is not a received mailer. You can use the `hasBeenReceived` field of the `SMPMailerState` structure (a parameter of the `SMPGetMailerState` function) to determine whether the top mailer has been received.

Use the `SMPAddMainEnclosure` function to add a main enclosure to the letter. The mailer does not display the contents of the letter's main enclosure in the Enclosures field.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

The enclosure is not actually added until well after this function has returned. Therefore, after calling the `SMPAddAttachment` function, you should call the `WaitNextEvent`

routine so that you yield time to the Finder to process Apple events while in the background. You must wait until the Standard Mail Package has finished copying the enclosure into the letter before you add anything else to the letter or try to send or save the letter. The `SMPMailerEvent` function uses the `kSMPCreateCopyWindowBit` and `kSMPDisposeCopyWindowBit` status bits to inform you of the progress of the copy operation.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$126E

RESULT CODES

`noErr`	0	No error
`fnfErr`	–43	File not found
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPMailerUneditable`	–1912	Mailer cannot be edited
`kSMPTooManyEnclosures`	–1928	More than 50 total files and folders

SEE ALSO

You can use the `SMPAttachDialog` function (described next) to display a dialog box that lets the user add an enclosure to a mailer.

You can use the `SMPGetMailerState` function (page 3-69) to determine if the top mailer is for a received letter.

You can use the `SMPGetListItemInfo` function (page 3-113) to list the enclosures that the user entered in the Enclosures field of a mailer.

Call the `SMPMailerEvent` function (page 3-63) to handle mailer events and to determine the status of the copy operation that occurs when you call the `SMPAddMainEnclosure` function.

Use the `SMPAddMainEnclosure` function (page 3-90) to add a main enclosure to a letter.

SMPAttachDialog

The `SMPAttachDialog` function displays a dialog box that lets the user add a disk file as an enclosure to a letter.

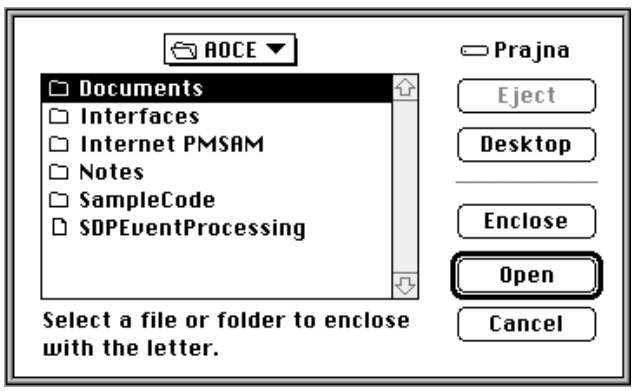
```
pascal OSErr SMPAttachDialog (WindowPtr window);
```

`window` The window containing the mailer.

DESCRIPTION

The Standard Mail Package provides a dialog box that lets a user add a disk file or folder as an enclosure to a letter (Figure 3-8). You can use the `SMPAttachDialog` function to display this same dialog box as an easy way to provide an Add Enclosures command. The `SMPAttachDialog` function adds enclosures only to the most recent mailer in the specified window, and then only if it is not a received mailer. You can use the `hasBeenReceived` field of the `SMPMailerState` structure (a parameter of the `SMPGetMailerState` function) to determine whether the top mailer is editable.

Figure 3-8 Add Enclosure dialog box



Use the `SMPAddMainEnclosure` function to add a main enclosure to the letter. The mailer does not display the contents of the letter's main enclosure in the Enclosures field.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

The enclosure is not actually added until well after this function has returned. Therefore, after calling the `SMPAttachDialog` function, you should call the `WaitNextEvent` routine so that you yield time to the Finder to process Apple events while in the background. You must wait until the Standard Mail Package has finished copying the enclosure into the letter before you add anything else to the letter or try to send or save the letter. The `SMPMailerEvent` function uses the `kSMPCreateCopyWindowBit` and `kSMPDisposeCopyWindowBit` status bits to inform you of the progress of the copy operation.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1276

RESULT CODES

`noErr`	0	No error
`userCanceledErr`	–128	User clicked Cancel button
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPMailerUneditable`	–1912	Mailer cannot be edited
`kSMPTooManyEnclosures`	–1928	More than 50 total files and folders

SEE ALSO

You can use the `SMPAddAttachment` function (page 3-119) if you want to provide your own interface that lets the user add an enclosure to a mailer.

You can use the `SMPGetMailerState` function (page 3-69) to determine if the top mailer is for a received letter and is therefore uneditable.

Call the `SMPMailerEvent` function (page 3-63) to handle mailer events and to determine the status of the copy operation that occurs when you call the `SMPAddMainEnclosure` function.

Use the `SMPAddMainEnclosure` function (page 3-90) to add a main enclosure to a letter.

Application-Defined Functions

This section describes the callback routines that you may provide for Standard Mail Package functions. Your `MyPrepareMailerForDrawing` routine restores your window port to a standard state so the Standard Mail Package can draw into it. Your `MyDrawImage` routine (page 3-123) images a document for the Standard Mail Package. The Standard Mail Package calls your `MyFrontWindowCB` routine (page 3-124) to determine which window is active when processing a key-down event.

MyPrepareMailerForDrawing

You may need to provide a `MyPrepareMailerForDrawing` routine to the `SMPNewMailer` function to make sure that the Standard Mail Package can draw a mailer in your window.

```
pascal void MyPrepareMailerForDrawing (WindowPtr window,
                                       long clientData);
```

`window` A pointer to the window into which the Standard Mail Package wants to draw.

`clientData`
Reserved for your use. You specify this value when you call the `SMPNewMailer` function, and that function passes the value unaltered to your callback routine.

DESCRIPTION

If you ever change the clip region, coordinates, or other aspects of your window's graphics port, you must provide a drawing-preparation routine that restores the window to its original state. The Standard Mail Package calls this routine before it draws into your window to add a new mailer or alter an existing mailer. You provide a pointer to your drawing-preparation routine when you call the `SMPNewMailer` function, the `SMPMailerReply` function, and the `SMPOpenLetter` function.

SPECIAL CONSIDERATIONS

You must make sure that your code for this routine is in a locked segment.

The `SMPNewMailer` function preserves your application's A5 world when it calls your drawing-preparation routine. Therefore, you have access to your application's global variables from this routine.

SEE ALSO

The `SMPNewMailer` function is described on page 3-46.

The `SMPMailerReply` function is described on page 3-51.

The `SMPOpenLetter` function is described on page 3-94.

MyDrawImage

The `MyDrawImage` function is a callback routine you must provide if you call the `SMPImage` function or if you specify `kSMPSendAsImageMask` for the `sendAs` field of the parameter block used by the `SMPSendLetter` function.

```
pascal void MyDrawImage (long refcon, Boolean inColor);
```

`refcon` A reference constant that you can use for your own purposes.

`inColor` A Boolean value that indicates whether the Standard Mail Package is providing a color graphics port to your image-drawing routine. This parameter is significant only in image-information structures passed to image-drawing routines.

DESCRIPTION

You provide a pointer to your image-drawing routine in the `drawImageProc` parameter when you call the `SMPImage` function and in the `drawImageProc` field of the parameter block when you call the `SMPSendLetter` function. Your image-drawing routine must call the `SMPNewPage` function before it draws each page of the document. You should call the `SMPImageErr` function rather than the `QDError` function after each QuickDraw routine you call. When you are finished imaging the document, just return.

If the user has color QuickDraw and you specified `true` for the `supportsColor` parameter of the `SMPImage` function or the `supportsColor` field of the parameter block used by `SMPSendLetter`, then the Standard Mail Package provides you with a color graphics port when it calls your image-drawing routine.

If you are imaging a letter that includes one or more mailers, you should image the mailers as cover pages before imaging the document. To do so, your image-drawing routine should first call the `SMPPrepareCoverPages` function to prepare the cover pages and to determine the total number of cover pages. Then for each cover page, you should call the `SMPNewPage` function and then the `SMPDrawNthCoverPage` function.

SPECIAL CONSIDERATIONS

If you change the graphics port within your image-drawing routine, you must change it back before calling the `SMPNewPage` or `SMPImageErr` functions.

SEE ALSO

The `SMPSendLetter` function is described on page 3-37. The `SMPImage` function is described on page 3-88.

You must call the `SMPNewPage` function (page 3-41) before you draw each page.

You should call the `SMPImageErr` function (page 3-41) after each QuickDraw routine you call.

To prepare cover pages for a mailer, you must call the `SMPPrepareCoverPages` function (page 3-107). To draw each cover page, you call the `SMPDrawNthCoverPage` function (page 3-108).

You can call the `GetPort` routine to determine the current graphics port. The `GetPort` routine is described in *Inside Macintosh: Imaging With QuickDraw*.

MyFrontWindowCB

The `MyFrontWindowCB` function is a callback routine you can provide with the `SMPMailerEvent` function. If you provide this function, the Standard Mail Package calls your `MyFrontWindowCB` function to determine which is the active window when processing a key-down event.

```
pascal WindowPtr MyFrontWindowCB (long clientData);
```

`clientData`

Reserved for your use. You specify this value when you call the `SMPMailerEvent` function, and that function passes the value unaltered to your callback routine.

DESCRIPTION

You can provide a pointer to your front-window routine when you call the `SMPMailerEvent` function. Your front-window routine returns a pointer to the window that you want the `SMPMailerEvent` function to treat as the frontmost window. You might use this callback routine, for example, if your application displays a status dialog box in front of your application's main window on the screen, but you want any key-down events to apply to your application's main window. If, as is the case with most applications, you do not have any windows in front of your main application window, specify `nil` for the `frontWindowCB` parameter of the `SMPMailerEvent` function. In that case the Standard Mail Package uses the Window Manager's `FrontWindow` routine to determine the frontmost window.

SPECIAL CONSIDERATIONS

The `SMPMailerEvent` function preserves your application's A5 world when it calls your front-window routine. Therefore, you have access to your application's global variables from this routine.

SEE ALSO

The `SMPMailerEvent` function is described on page 3-63.

MySendOptionsFilterProc

The send-options filter procedure is a routine you can provide when you call the `SMPSendOptionsDialog` function. This routine extends the send-options dialog box.

```
pascal Boolean MySendOptionsFilterProc (DialogPtr theDialog,
                                        EventRecord* theEvent,
                                        short itemHit,
                                        long clientData);
```

`theDialog` A pointer to the dialog structure for the send-options dialog box.

`theEvent` The event that was just received by the send-options dialog box.

`itemHit` If the dialog box has just received a mouse-down event, this parameter indicates the number of the dialog item in which the mouse-down event occurred.

`clientData`
A constant reserved for your use. You specify this value when you call the `SMPSendOptionsDialog` function.

DESCRIPTION

If you provide a filter routine when you call the `SMPSendOptionsDialog` function, the Standard Mail Package calls your filter routine each time it receives an event for the send-options dialog box. If your filter routine returns `true`, the Standard Mail Package assumes you handled the event. If your filter routine returns `false`, the Standard Mail Package handles the event normally. You can alter the event before returning `false`.

To allow your filter routine to add new items to the send-options dialog box and to clean up before it removes the dialog box, the Standard Mail Package sends your function two pseudoevents:

```
enum {
   kSMPSendOptionsStart       = -1,
   kSMPSendOptionsEnd         = -2
};
```

When your filter routine receives the `kSMPSendOptionsStart` event, you can call the `CountDITL` routine to determine the number of items already in the dialog box. You can then call the `AppendDITL` function to add new items to the dialog box, as follows:

```
AppendDITL(theDialog, myDITL, appendDITLBottom)
```

The parameter `myDITL` describes the new items you wish to add to the dialog box. When you begin numbering new items, increment by 1 the number returned by the `CountDITL` routine. When your filter routine receives the `kSMPSendOptionsStart` event, you can also allocate memory, initialize menus, and so forth.

Immediately before closing the send-options dialog box, the Standard Mail Package sends a `kSMPSendOptionsEnd` event to your filter routine. You should then deallocate any memory that you allocated earlier.

SPECIAL CONSIDERATIONS

Do not make any assumptions about the number or position of the standard items in the send-options dialog box, as Apple Computer, Inc., reserves the right to change this dialog box at any time.

SEE ALSO

The `SMPSendOptionsDialog` function is described on page 3-73.

Summary of the Standard Mail Package

C Summary

Constants and Data Types

```
#define gestaltSMPMailerVersion'malr'
#define gestaltSMPSPSendLetterVersion'spsl'
#define kSMPNativeFormatName'natv'

#define typeLetterSpec'lttr'

/* wildcard used for filtering letter types */

enum {
   FilterAnyLetter='ltr*',
   FilterAppleLetterContent='ltc*',
   FilterImageContent='lti*'
};

/* SMPPSendAs values.  You may add the following values together to
determine how the file is sent, but you may not set both
kSMPSendAsEnclosureMask and kSMPSendFileOnlyMask. */

enum {
   kSMPSendAsEnclosureBit, /* appears as letter with enclosures */
   kSMPSendFileOnlyBit,    /* appears as a file in mailbo. */
   kSMPSendAsImageBit      /* letter includes image of content */
};

/* values of SMPPSendAs */

enum {
   kSMPSendAsEnclosureMask = 1<<kSMPSendAsEnclosureBit,
   kSMPSendFileOnlyMask    = 1<<kSMPSendFileOnlyBit,
   kSMPSendAsImageMask     = 1<<kSMPSendAsImageBit
};

typedef Byte SMPPSendAs;
```

```
enum {
   kSMPAppMustHandleEventBit,
   kSMPAppShouldIgnoreEventBit,
   kSMPContractedBit,
   kSMPExpandedBit,
   kSMPMailerBecomesTargetBit,
   kSMPAppBecomesTargetBit,
   kSMPCursorOverMailerBit,
   kSMPCreateCopyWindowBit,
   kSMPDisposeCopyWindowBit
};

/* values of SMPMailerResult */

enum {
   kSMPAppMustHandleEventMask    = 1<<kSMPAppMustHandleEventBit,
   kSMPAppShouldIgnoreEventMask  = 1<<kSMPAppShouldIgnoreEventBit,
   kSMPContractedMask            = 1<<kSMPContractedBit,
   kSMPExpandedMask              = 1<<kSMPExpandedBit,
   kSMPMailerBecomesTargetMask   = 1<<kSMPMailerBecomesTargetBit,
   kSMPAppBecomesTargetMask      = 1<<kSMPAppBecomesTargetBit,
   kSMPCursorOverMailerMask      = 1<<kSMPCursorOverMailerBit,
   kSMPCreateCopyWindowMask      = 1<<kSMPCreateCopyWindowBit,
   kSMPDisposeCopyWindowMask     = 1<<kSMPDisposeCopyWindowBit
};

typedef unsigned long SMPMailerResult;

/* values of SMPMailerComponent*/

enum {
   kSMPOther         =  -1,
   kSMPFrom          =  32,
   kSMPTo            =  20,
   kSMPRegarding     =  22,
   kSMPSendDateTime  =  29,
   kSMPAttachments   =  26,
   kSMPAddressOMatic =  16
};

typedef unsigned long SMPMailerComponent;
```

```
enum {
   kSMPToAddress  =  kMailToBit,
   kSMPCCAddress  =  kMailCcBit,
   kSMPBCCAddress =  kMailBccBit
};

typedef MailAttributeID SMPAddressType;

enum {
   kSMPUndoCommand,
   kSMPCutCommand,
   kSMPCopyCommand,
   kSMPPasteCommand,
   kSMPClearCommand,
   kSMPSelectAllCommand
};

typedef unsigned short SMPEditCommand;

enum {
   kSMPUndoDisabled,
   kSMPAppMayUndo,
   kSMPMailerUndo
};

typedef unsigned short SMPUndoState;

/* SMPSendFormatMask: Bitfield indicating which combinations of formats are
included in, should be included in, or can be included in a letter. */

enum {
   kSMPNativeBit,
   kSMPImageBit,
   kSMPStandardInterchangeBit
};

/* values of SMPSendFormatMask */

enum {
   kSMPNativeMask               = 1<<kSMPNativeBit,
   kSMPImageMask                = 1<<kSMPImageBit,
   kSMPStandardInterchangeMask  = 1<<kSMPStandardInterchangeBit
};

typedef unsigned long SMPSendFormatMask;
```

```
/* pseudo-events passed to the client's filter proc for initialization and
cleanup */

enum {
   kSMPSendOptionsStart = -1,
   kSMPSendOptionsEnd   = -2
};

enum {
   kSMPSave,
   kSMPSaveAs,
   kSMPSaveACopy
};

typedef unsigned short SMPSaveType;

/* values of MailSegmentType */
enum {
   kMailInvalidSegmentType    = 0,
   kMailTextSegmentType       = 1,
   kMailPictSegmentType       = 2,
   kMailSoundSegmentType      = 3,
   kMailStyledTextSegmentType = 4,
   kMailMovieSegmentType      = 5
};

typedef unsigned short MailSegmentType;

/* values of MailBlockMode */
enum {
   kMailFromStart = 1,     /* offset calculated from start of block */
   kMailFromLEOB  = 2,     /* offset calculated from end of block */
   kMailFromMark  = 3      /* offset calculated from current mark */
};

typedef short MailBlockMode;

struct SMPRecipientDescriptor
{
   struct SMPRecipientDescriptor *next;      /* pointer to next element */
   OSErr                         result;     /* result code */
   OCEPackedRecipient            *recipient; /* packed recipient address */
   unsigned long                 size;       /* size of recipient address */
```

```
   MailRecipient                theAddress; /* unpacked recipient address */
   RecordID                     theRID;     /* record ID of recipient */
};

typedef struct SMPRecipientDescriptor SMPRecipientDescriptor;
typedef SMPRecipientDescriptor *SMPRecipientDescriptorPtr;

struct SMPEnclosureDescriptor
{
   struct SMPEnclosureDescriptor *next;         /* pointer to next element */
   OSErr                         result;        /* result code */
   FSSpec                        fileSpec;      /* file specifier of
                                                   enclosure */
   OSType                        fileCreator;   /* creator of enclosure */
   OSType                        fileType;      /* file type of enclosure */
};

typedef struct SMPEnclosureDescriptor SMPEnclosureDescriptor;
typedef SMPEnclosureDescriptor *SMPEnclosureDescriptorPtr;

struct LetterDescriptor {
   Boolean onDisk;
   union {
      FSSpec fileSpec;
      LetterSpec mailboxSpec;
   }u;
};

typedef struct LetterDescriptor LetterDescriptor;

struct SMPLetterPB
{
   OSErr                      result;        /* function result */
   RStringPtr                 subject;       /* subject of letter */
   AuthIdentity               senderIdentity;/* identity of sender */
   SMPRecipientDescriptorPtr  toList;        /* list of addressees */
   SMPRecipientDescriptorPtr  ccList;        /* list of cc addressees */
   SMPRecipientDescriptorPtr  bccList;       /* list of bcc addressees */
   ScriptCode                 script;        /* script code for language */
   Size                       textSize;      /* length of body data */
   Ptr                        textBuffer;    /* body of the letter */
   SMPPSendAs                 sendAs;        /* file, enclosure, or image */
   Byte                       padByte;       /* reserved */
   SMPEnclosureDescriptorPtr  enclosures;    /* files to be enclosed */
   SMPDrawImageProcPtr        drawImageProc; /* your imaging routine */
```

```
   long                        imageRefCon;   /* for your use */
   Boolean                     supportsColor; /* true for a color grafPort */
};

typedef struct SMPLetterPB SMPLetterPB;
typedef SMPLetterPB *SMPLetterPBPtr;

struct SMPCloseOptions {
   Boolean              moveToTrash;
   Boolean              addTag;
   RString32            tag;
};

typedef struct SMPCloseOptions SMPCloseOptions;
typedef SMPCloseOptions *SMPCloseOptionsPtr;

struct SMPMailerState {
   short                mailerCount;
   short                currentMailer;
   Point                upperLeft;
   Boolean              hasBeenReceived;
   Boolean              isTarget;
   Boolean              isExpanded;
   Boolean              canMoveToTrash;
   Boolean              canTag;
   Byte                 padByte2;
   unsigned long        changeCount;
   SMPMailerComponent   targetComponent;
   Boolean              canCut;
   Boolean              canCopy;
   Boolean              canPaste;
   Boolean              canClear;
   Boolean              canSelectAll;
   Byte                 padByte3;
   SMPUndoState         undoState;
   Str63                undoWhat;
};

typedef struct SMPMailerState SMPMailerState;

struct SMPSendOptions {
   Boolean     signWhenSent;
   IPMPriority priority;
};
```

```
typedef struct SMPSendOptions SMPSendOptions;
typedef SMPSendOptions *SMPSendOptionsPtr, **SMPSendOptionsHandle;

/* SMPSendFormat: Structure describing the format of a letter.  If
kSMPNativeMask bit is set, the whichNativeFormat field indicates which of
the client-defined formats to use. */

struct SMPSendFormat {
   SMPSendFormatMask whichFormats;
   short whichNativeFormat;         /* zero-based */
};

typedef struct SMPSendFormat SMPSendFormat;

struct LetterSpec
{
   unsigned long     spec[3];
};

struct SMPLetterInfo {
      OSType      letterCreator;
      OSType      letterType;
      RString32   subject;
      RString32   sender;
};

typedef struct SMPLetterInfo SMPLetterInfo;

typedef struct MailTime {
   UTCTime     time;               /* current UTC (GMT) time */
   UTCOffset   offset;             /* in seconds from GMT */
};

typedef struct MailTime MailTime;

typedef  unsigned long  UTCTime;    /* seconds since 1/1/1904 */
typedef  long           UTCOffset;  /* correct for local time */

/* pointers to functions for application-defined callback functions */

typedef pascal void (*SMPDrawImageProcPtr)(long refcon, Boolean inColor);

typedef pascal WindowPtr (*FrontWindowProcPtr) (long clientData);

typedef pascal void (*PrepareMailerForDrawingProcPtr) (WindowPtr window,
                                                       long clientData);
```

```
typedef pascal Boolean (*SendOptionsFilterProc) (DialogPtr theDialog,
                                                 EventRecord* theEvent,
                                                 short itemHit,
                                                 long clientData);
```

Standard Mail Package Functions

Send-Letter Functions

```
pascal OSErr SMPSendLetter  (SMPLetterPBPtr theLetter);
pascal OSErr SMPNewPage     (OpenCPicParams *newHeader);
pascal OSErr SMPImageErr    (void);
pascal OSErr SMPResolveToRecipient
                            (PackedDSSpecPtr dsSpec,
                             SMPRecipientDescriptorPtr *recipientList,
                             AuthIdentity identity);
```

Providing Mailers in Your Windows

```
pascal OSErr SMPInitMailer  (long mailerVersion);
pascal OSErr SMPNewMailer   (WindowPtr window,
                             Point upperLeft,
                             Boolean canContract,
                             Boolean initiallyExpanded,
                             AuthIdentity identity,
                             const PrepareMailerForDrawingProcPtr
                              prepareMailerForDrawingCB,
                             long clientData);
pascal OSErr SMPGetDimensions
                            (short *width,
                             short *contractedHeight,
                             short *expandedHeight);
pascal OSErr SMPMailerForward
                            (WindowPtr window,
                             AuthIdentity from);
```

```
pascal OSErr SMPMailerReply
                            (WindowPtr originalLetter,
                             WindowPtr newLetter,
                             Boolean replyToAll,
                             Point upperLeft,
                             Boolean canContract,
                             Boolean initiallyExpanded,
                             AuthIdentity identity,
                             const PrepareMailerForDrawingProcPtr
                             prepareMailerForDrawingCB,
                             long clientData);
pascal OSErr SMPGetTabInfo  (SMPMailerComponent *firstTab,
                             SMPMailerComponent *lastTab);
pascal OSErr SMPBecomeTarget
                            (WindowPtr window,
                             Boolean becomeTarget,
                             SMPMailerComponent whichField);
pascal OSErr SMPExpandOrContract
                            (WindowPtr window,
                             Boolean expand);
pascal OSErr SMPMoveMailer  (WindowPtr window,
                             short dh,
                             short dv);
pascal OSErr SMPTagDialog   (WindowPtr window,
                             RString32 *theTag);
pascal OSErr SMPPrepareToClose
                            (WindowPtr window);
pascal OSErr SMPCloseOptionsDialog
                            (WindowPtr window,
                             SMPCloseOptionsPtr closeOptions);
pascal OSErr SMPDisposeMailer
                            (WindowPtr window,
                             SMPCloseOptionsPtr closeOptions);
```

Handling Events in Mailers

```
pascal OSErr SMPMailerEvent
                            (const EventRecord *event,
                             SMPMailerResult *whatHappened,
                             const FrontWindowProcPtr frontWindowCB,
                             long clientData);
pascal OSErr SMPMailerEditCommand
                            (WindowPtr window,
                             SMPEditCommand command,
                             SMPMailerResult *whatHappened);
```

```
pascal OSErr SMPGetMailerState
                            (windowPtr window,
                             SMPMailerState *itsState);
pascal OSErr SMPClearUndo   (WindowPtr window);
pascal OSErr SMPDrawMailer  (WindowPtr window);
```

Sending and Saving Mail

```
pascal OSErr SMPSendOptionsDialog
                            (WindowPtr window,
                             Str255 documentName,
                             StringPtr nativeFormatNames[],
                             unsigned short nameCount,
                             SMPSendFormatMask canSend,
                             SMPSendFormat *currentFormat,
                             SendOptionsFilterProc filterProc,
                             long clientData,
                             SMPSendFormat *shouldSend,
                             SMPSendOptionsPtr sendOptions);
pascal OSErr SMPContentChanged
                            (WindowPtr window);
pascal OSErr SMPBeginSave   (WindowPtr window,
                             const FSSpec *diskLetter,
                             OSType creator,
                             OSType filetype,
                             SMPSaveType saveType,
                             Boolean *mustAddContent);
pascal OSErr SMPEndSave     (WindowPtr window,
                             Boolean okToSave);
pascal OSErr SMPBeginSend   (WindowPtr window,
                             OSType creator,
                             OSType fileType,
                             SMPSendOptionsPtr sendOptions,
                             Boolean *mustAddContent);
pascal OSErr SMPPrepareToChange
                            (WindowPtr window);
pascal OSErr SMPEndSend     (WindowPtr window,
                             Boolean okToSend);
```

```
pascal OSErr SMPAddContent  (WindowPtr window,
                             MailSegmentType segmentType,
                             Boolean appendFlag,
                             void *buffer,
                             unsigned long bufferSize,
                             StScrpRec *textScrap,
                             Boolean startNewScript,
                             ScriptCode script);
pascal OSErr SMPImage       (WindowPtr window,
                             SMPDrawImageProcPtr drawImageProc,
                             long imageRefCon,
                             Boolean supportsColor);
pascal OSErr SMPAddMainEnclosure
                            (WindowPtr window,
                             const FSSpec *enclosure);
pascal OSErr SMPAddBlock    (WindowPtr window,
                             const OCECreatorType *blockType,
                             Boolean append,
                             void *buffer,
                             unsigned long bufferSize,
                             MailBlockMode mode,
                             unsigned long offset);
```

Reading Mail

```
pascal OSErr SMPGetLetterInfo
                            (LetterSpec *mailboxSpec,
                             SMPLetterInfo *info);
pascal OSErr SMPOpenLetter  (const LetterDescriptor *letter,
                             WindowPtr window,
                             Point upperLeft,
                             Boolean canContract,
                             Boolean initiallyExpanded,
                             const PrepareMailerForDrawingProcPtr
                              prepareMailerForDrawingCB,
                             long clientData);
pascal OSErr SMPGetNextLetter
                            (OSType *typesList,
                             short numTypes,
                             LetterDescriptor *adjacentLetter);
```

```
pascal OSErr SMPReadContent
                            (WindowPtr window,
                             MailSegmentMask segmentTypeMask,
                             void *buffer,
                             unsigned long bufferSize,
                             unsigned long *dataSize,
                             StScrpRec *textScrap,
                             ScriptCode *script,
                             MailSegmentType *segmentType,
                             Boolean *endOfScript,
                             Boolean *endOfSegment,
                             Boolean *endOfContent,
                             long *segmentLength,
                             long *segmentID);
pascal OSErr SMPGetFontNameFromLetter
                            (WindowPtr window,
                             short fontNum,
                             str255 fontName,
                             Boolean doneWithFontTable);
pascal OSErr SMPGetMainEnclosureFSSpec
                            (WindowPtr window,
                             FSSpec *enclosureDir);
pascal OSErr SMPEnumerateBlocks
                            (WindowPtr window,
                             unsigned short startIndex,
                             void *buffer,
                             unsigned long bufferSize,
                             unsigned long *dataSize,
                             unsigned short *nextIndex,
                             Boolean *more);
pascal OSErr SMPReadBlock   (WindowPtr window,
                             const OCECreatorType *blockType,
                             unsigned short blockIndex,
                             void *buffer,
                             unsigned long bufferSize,
                             unsigned long dataOffset,
                             unsigned long *dataSize,
                             Boolean *endOfBlock,
                             unsigned long *remaining);
```

Printing Mailers

```
pascal OSErr SMPPrepareCoverPages
                            (windowPtr window,
                             short *pageCount);
```

```
pascal OSErr SMPDrawNthCoverPage
                            (WindowPtr window,
                             short pageNumber,
                             Boolean doneDrawingCoverPages);
```

Getting and Setting Information in the Mailer

```
pascal OSErr SMPGetComponentSize
                            (WindowPtr window,
                             unsigned short whichMailer,
                             SMPMailerComponent whichField,
                             unsigned short *size);
pascal OSErr SMPGetComponentInfo
                            (WindowPtr window,
                             unsigned short whichMailer,
                             SMPMailerComponent whichField,
                             void *buffer);
pascal OSErr SMPGetListItemInfo
                            (WindowPtr window,
                             unsigned short whichMailer,
                             SMPMailerComponent whichField,
                             void *buffer,
                             unsigned long bufferLength,
                             unsigned short startItem,
                             unsigned short *itemCount,
                             unsigned short *nextItem,
                             Boolean *more);
pascal OSErr SMPSetSubject  (WindowPtr window,
                             const RString *text);
pascal OSErr SMPSetFromIdentity
                            (WindowPtr window,
                             AuthIdentity from);
pascal OSErr SMPAddAddress  (WindowPtr window,
                             SMPAddressType addrType,
                             OCEPackedRecipient *address);
pascal OSErr SMPAddAttachment
                            (WindowPtr window,
                             const FSSpec *attachment);
pascal OSErr SMPAttachDialog
                            (WindowPtr window);
```

Application-Defined Functions

```
pascal void MyPrepareMailerForDrawing
                            (WindowPtr window,
                             long clientData);
pascal void MyDrawImage     (long refcon, Boolean inColor);
pascal WindowPtr MyFrontWindowCB
                            (long clientData);
pascal Boolean MySendOptionsFilterProc
                            (DialogPtr theDialog,
                             EventRecord* theEvent,
                             short itemHit,
                             long clientData);
```

Pascal Summary

Constants

```
CONST
gestaltSMPMailerVersion       = 'malr';
gestaltSMPSPSendLetterVersion = 'spsl';
kSMPNativeFormatName          = 'natv';

typeLetterSpec                = 'lttr';

{ wildcard used for filtering letter types }

FilterAnyLetter               =  'ltr*';
FilterAppleLetterContent      =  'ltc*';
FilterImageContent            =  'lti*';

{ SMPPSendAs values.  You may add the following values together to determine
how the file is sent, but you may not set both kSMPSendAsEnclosureMask and
kSMPSendFileOnlyMask. }

kSMPSendAsEnclosureBit  = 0;     { appears as letter with enclosures }
kSMPSendFileOnlyBit     = 1;     { appears as a file in mailbox }
kSMPSendAsImageBit      = 2;     { letter includes image of content }
```

```
{ values of SMPPSendAs }
kSMPSendAsEnclosureMask    = $01;  {1<<kSMPSendAsEnclosureBit}
kSMPSendFileOnlyMask       = $02;  {1<<kSMPSendFileOnlyBit}
kSMPSendAsImageMask        = $04;  {1<<kSMPSendAsImageBit}

kSMPAppMustHandleEventBit     = 0;
kSMPAppShouldIgnoreEventBit   = 1;
kSMPContractedBit             = 2;
kSMPExpandedBit               = 3;
kSMPMailerBecomesTargetBit    = 4;
kSMPAppBecomesTargetBit       = 5;
kSMPCursorOverMailerBit       = 6;
kSMPCreateCopyWindowBit       = 7;
kSMPDisposeCopyWindowBit      = 8;

{ values of SMPMailerResult }
kSMPAppMustHandleEventMask    = $00000001; {1<<kSMPAppMustHandleEventBit}
kSMPAppShouldIgnoreEventMask  = $00000002; {1<<kSMPAppShouldIgnoreEventBit}
kSMPContractedMask            = $00000004; {1<<kSMPContractedBit}
kSMPExpandedMask              = $00000008; {1<<kSMPExpandedBit}
kSMPMailerBecomesTargetMask   = $00000010; {1<<kSMPMailerBecomesTargetBit}
kSMPAppBecomesTargetMask      = $00000020; {1<<kSMPAppBecomesTargetBit}
kSMPCursorOverMailerMask      = $00000040; {1<<kSMPCursorOverMailerBit}
kSMPCreateCopyWindowMask      = $00000080; {1<<kSMPCreateCopyWindowBit}
kSMPDisposeCopyWindowMask     = $00000100; {1<<kSMPDisposeCopyWindowBit}

{ values of SMPMailerComponent }
kSMPOther            =  -1;
kSMPFrom             =  32;
kSMPTo               =  20;
kSMPRegarding        =  22;
kSMPSendDateTime     =  29;
kSMPAttachments      =  26;
kSMPAddressOMatic    =  16;

kSMPToAddress  =  kMailToBit;
kSMPCCAddress  =  kMailCcBit;
kSMPBCCAddress =  kMailBccBit;

kSMPUndoCommand = 0;
kSMPCutCommand = 1;
kSMPCopyCommand = 2;
```

```
kSMPPasteCommand = 3;
kSMPClearCommand = 4;
kSMPSelectAllCommand = 5;

kSMPUndoDisabled = 0;
kSMPAppMayUndo = 1;
kSMPMailerUndo = 2;

{ SMPSendFormatMask: Bitfield indicating which combinations of formats are
included in, should be included in, or can be included in a letter. }

kSMPNativeBit = 0;
kSMPImageBit = 1;
kSMPStandardInterchangeBit = 2;

{ values of SMPSendFormatMask }
kSMPNativeMask               = $00000001;  {1<<kSMPNativeBit}
kSMPImageMask                = $00000002;  {1<<kSMPImageBit}
kSMPStandardInterchangeMask  = $00000004;  {1<<kSMPStandardInterchangeBit}

{ pseudo-events passed to the client's filter proc for initialization and
cleanup }
kSMPSendOptionsStart= -1;
kSMPSendOptionsEnd= -2;

kSMPSave = 0;
kSMPSaveAs = 1;
kSMPSaveACopy = 2;

{ values of MailSegmentType }
kMailInvalidSegmentType= 0;
kMailTextSegmentType= 1;
kMailPictSegmentType= 2;
kMailSoundSegmentType= 3;
kMailStyledTextSegmentType= 4;
kMailMovieSegmentType= 5;

{ values of MailBlockMode }
kMailFromStart= 1;{ offset calculated from start of block }
kMailFromLEOB= 2; { offset calculated from end of block }
kMailFromMark= 3; { offset calculated from the current mark }
```

Data Types

```
TYPE
SMPSendFormatMask = LONGINT;

SMPSaveType = INTEGER;

SMPMailerResult = LONGINT;

SMPMailerComponent = LONGINT;

SMPAddressType = MailAttributeID;

SMPEditCommand = INTEGER;

SMPUndoState = INTEGER;

SMPPSendAs = Byte;

MailBlockMode = INTEGER;

MailSegmentType = INTEGER;

SMPRecipientDescriptor = RECORD
   next:          ^SMPRecipientDescriptor;   { pointer to next element }
   result:        OSErr;                     { result code }
   recipient:     ^OCEPackedRecipient;       { packed recipient address }
   size:          LONGINT;                   { size of recipient address }
   theAddress:    MailRecipient;             { unpacked recipient address }
   theRID:        RecordID;                  { record ID of recipient }
   END;

SMPRecipientDescriptorPtr = ^SMPRecipientDescriptor;

SMPEnclosureDescriptor = RECORD
   next:          ^SMPEnclosureDescriptor;   { pointer to next element }
   result:        OSErr;                     { result code }
   fileSpec:      FSSpec;                    { file specifier of enclosure }
   fileCreator:   OSType;                    { creator of enclosure }
   fileType:      OSType;                    { file type of enclosure }
   END;

SMPEnclosureDescriptorPtr = ^SMPEnclosureDescriptor;
```

```
LetterDescriptor = RECORD
   onDisk: BOOLEAN;
   CASE INTEGER OF
      1: (fileSpec: FSSpec);
      2: (mailboxSpec: LetterSpec);
   END;

SMPLetterPB = PACKED RECORD
   result:        OSErr;                    { function result }
   subject:       RStringPtr;               { subject of letter }
   senderIdentity:AuthIdentity;             { identity of sender }
   toList:        SMPRecipientDescriptorPtr; { list of addressees }
   ccList:        SMPRecipientDescriptorPtr; { list of cc addressees }
   bccList:       SMPRecipientDescriptorPtr; { list of bcc addressees }
   script:        ScriptCode;               { script code for language }
   textSize:      Size;                     { length of body data }
   textBuffer:    Ptr;                      { body of the letter }
   sendAs:        SMPPSendAs;               { letter,enclosure, or image }
   padByte:       Byte;                     { reserved }
   enclosures:    SMPEnclosureDescriptorPtr; { files to be enclosed }
   drawImageProc: SMPDrawImageProcPtr;      { your imaging routine }
   imageRefCon:   LONGINT;                  { for your use }
   supportsColor: BOOLEAN;                  { true for a color grafPort }
   END;

SMPLetterPBPtr = ^SMPLetterPB;

SMPCloseOptions = RECORD
   moveToTrash: BOOLEAN;
   addTag: BOOLEAN;
   tag: RString32;
   END;

SMPCloseOptionsPtr = ^SMPCloseOptions;

SMPMailerState = RECORD
   mailerCount: INTEGER;
   currentMailer: INTEGER;
   upperLeft: Point;
   hasBeenReceived: BOOLEAN;
   isTarget: BOOLEAN;
   isExpanded: BOOLEAN;
   canMoveToTrash: BOOLEAN;
   canTag: BOOLEAN;
```

```
   {padByte2: Byte;}
   changeCount: LONGINT;
   targetComponent: SMPMailerComponent;
   canCut: BOOLEAN;
   canCopy: BOOLEAN;
   canPaste: BOOLEAN;
   canClear: BOOLEAN;
   canSelectAll: BOOLEAN;
   {padByte3: Byte;}
   undoState: SMPUndoState;
   undoWhat: Str63;
   END;

SMPSendOptions = RECORD
   signWhenSent: BOOLEAN;
   priority: IPMPriority;
   END;

SMPSendOptionsPtr = ^SMPSendOptions;
SMPSendOptionsHandle = ^SMPSendOptionsPtr;

{ SMPSendFormat: Structure describing the format of a letter.  If
kSMPNativeMask bit is set, the whichNativeFormat field indicates which of
the client-defined formats to use. }

SMPSendFormat = RECORD
   whichFormats: SMPSendFormatMask;
   whichNativeFormat: INTEGER;{ 0 based }
   END;

LetterSpec = RECORD
   spec: ARRAY[1..3] OF LONGINT;
   END;

SMPLetterInfo = RECORD
   letterCreator: OSType;
   letterType: OSType;
   subject: RString32;
   sender: RString32;
   END;
```

```
MailTime = RECORD
   time: UTCTime;    { current UTC (GMT) time }
   offset: UTCOffset;{ in seconds from GMT (positive is east) }
   END;

UTCTime = LONGINT;   { seconds since 1/1/1904 }
UTCOffset = LONGINT; { correct for local time }

{ pointers to functions for application-defined callback functions }

SMPDrawImageProcPtr = ProcPtr;
   { FUNCTION SMPDrawImageProcPtr(refcon: LONGINT; inColor: BOOLEAN): void;}

FrontWindowProcPtr = ProcPtr;
   { FUNCTION FrontWindowProcPtr(clientData: LONGINT): WindowPtr;}

PrepareMailerForDrawingProcPtr = ProcPtr;
   { FUNCTION PrepareMailerForDrawingProcPtr(window: WindowPtr;
                                        clientData: LONGINT): void;}

SendOptionsFilterProc = ProcPtr;
   { FUNCTION SendOptionsFilterProc(theDialog: DialogPtr;
                                VAR theEvent: EventRecord;
                                itemHit: INTEGER;
                                clientData: LONGINT): BOOLEAN;}
```

Standard Mail Package Functions

Send-Letter Functions

```
FUNCTION SMPSendLetter      (theLetter: SMPLetterPBPtr): OSErr;
FUNCTION SMPNewPage         (VAR newHeader: OpenCPicParams): OSErr;
FUNCTION SMPImageErr: OSErr;
FUNCTION SMPResolveToRecipient
                            (dsSpec: PackedDSSpecPtr;
                             VAR recipientList: SMPRecipientDescriptorPtr;
                             identity: AuthIdentity): OSErr;
```

Providing Mailers in Your Windows

```
FUNCTION SMPInitMailer      (mailerVersion: LONGINT): OSErr;
FUNCTION SMPNewMailer       (window: WindowPtr; upperLeft: Point;
                             canContract: BOOLEAN;
                             initiallyExpanded: BOOLEAN;
                             identity: AuthIdentity;
                             prepareMailerForDrawingCB:
                             PrepareMailerForDrawingProcPtr;
                             clientData: LONGINT): OSErr;
FUNCTION SMPGetDimensions   (VAR width: INTEGER; VAR contractedHeight:
                             INTEGER; VAR expandedHeight: INTEGER): OSErr;
FUNCTION SMPMailerForward   (window: WindowPtr; from: AuthIdentity): OSErr;
FUNCTION SMPMailerReply     (originalLetter: WindowPtr; newLetter:
                             WindowPtr; replyToAll: BOOLEAN; upperLeft:
                             Point; canContract: BOOLEAN;
                             initiallyExpanded: BOOLEAN;
                             identity: AuthIdentity;
                             prepareMailerForDrawingCB:
                             PrepareMailerForDrawingProcPtr;
                             clientData: LONGINT): OSErr;
FUNCTION SMPGetTabInfo      (VAR firstTab: SMPMailerComponent;
                             VAR lastTab: SMPMailerComponent): OSErr;
FUNCTION SMPBecomeTarget    (window: WindowPtr; becomeTarget: BOOLEAN;
                             whichField: SMPMailerComponent): OSErr;
FUNCTION SMPExpandOrContract
                            (window: WindowPtr; expand: BOOLEAN): OSErr;
FUNCTION SMPMoveMailer      (window: WindowPtr; dh: INTEGER; dv: INTEGER):
                             OSErr;
FUNCTION SMPTagDialog       (window: WindowPtr; theTag: RString32Ptr):
                             OSErr;
FUNCTION SMPPrepareToClose  (window: WindowPtr): OSErr;
FUNCTION SMPCloseOptionsDialog
                            (window: WindowPtr;
                             closeOptions: SMPCloseOptionsPtr): OSErr;
FUNCTION SMPDisposeMailer   (window: WindowPtr;
                             closeOptions: SMPCloseOptionsPtr): OSErr;
```

Handling Events in Mailers

```
FUNCTION SMPMailerEvent     (event: EventRecord;
                             VAR whatHappened: SMPMailerResult;
                             frontWindowCB: FrontWindowProcPtr;
                             clientData: LONGINT): OSErr;
FUNCTION SMPMailerEditCommand
                            (window: WindowPtr; command: SMPEditCommand;
                             VAR whatHappened: SMPMailerResult): OSErr;
FUNCTION SMPGetMailerState  (window: WindowPtr; VAR itsState:
                             SMPMailerState): OSErr;
FUNCTION SMPClearUndo       (window: WindowPtr): OSErr;
FUNCTION SMPDrawMailer      (window: WindowPtr): OSErr;
```

Sending and Saving Mail

```
FUNCTION SMPSendOptionsDialog
                            (window: WindowPtr; documentName: Str255;
                             VAR nativeFormatNames: StringPtr;
                             nameCount: INTEGER;
                             canSend: SMPSendFormatMask;
                             VAR currentFormat: SMPSendFormat;
                             filterProc: SendOptionsFilterProc;
                             clientData: LONGINT;
                             VAR shouldSend: SMPSendFormat;
                             sendOptions: SMPSendOptionsPtr): OSErr;
FUNCTION SMPContentChanged
                            (window: WindowPtr): OSErr;
FUNCTION SMPBeginSave       (window: WindowPtr; diskLetter: FSSpec;
                             creator: OSType; fileType: OSType;
                             saveType: SMPSaveType;
                             VAR mustAddContent: BOOLEAN): OSErr;
FUNCTION SMPEndSave         (window: WindowPtr; okToSave: BOOLEAN): OSErr;
FUNCTION SMPBeginSend       (window: WindowPtr; creator: OSType; fileType:
                             OSType; sendOptions: SMPSendOptionsPtr;
                             VAR mustAddContent: BOOLEAN): OSErr;
FUNCTION SMPPrepareToChange
                            (window: WindowPtr): OSErr;
FUNCTION SMPEndSend         (window: WindowPtr; okToSend: BOOLEAN): OSErr;
```

```
FUNCTION SMPAddContent      (window: WindowPtr; segmentType:
                             MailSegmentType; appendFlag: BOOLEAN; buffer:
                             UNIV Ptr; bufferSize: LONGINT; textScrap:
                             StScrpPtr; startNewScript: BOOLEAN; script:
                             ScriptCode): OSErr;
FUNCTION SMPImage           (window: WindowPtr; drawImageProc:
                             SMPDrawImageProcPtr; imageRefCon: LONGINT;
                             supportsColor: BOOLEAN): OSErr;
FUNCTION SMPAddMainEnclosure
                            (window: WindowPtr; enclosure: FSSpec): OSErr;
FUNCTION SMPAddBlock        (window: WindowPtr; blockType: OCECreatorType;
                             append: BOOLEAN; buffer: UNIV Ptr;
                             bufferSize: LONGINT; mode: MailBlockMode;
                             offset: LONGINT): OSErr;
```

Reading Mail

```
FUNCTION SMPGetLetterInfo   (VAR mailboxSpec: LetterSpec;
                             VAR info: SMPLetterInfo): OSErr;
FUNCTION SMPOpenLetter      (letter: LetterDescriptor; window: WindowPtr;
                             upperLeft: Point; canContract: BOOLEAN;
                             initiallyExpanded: BOOLEAN;
                             prepareMailerForDrawingCB:
                             PrepareMailerForDrawingProcPtr;
                             clientData: LONGINT): OSErr;
FUNCTION SMPGetNextLetter   (VAR typesList: OSType; numTypes: INTEGER;
                             VAR adjacentLetter: LetterDescriptor): OSErr;
FUNCTION SMPReadContent     (window: WindowPtr; segmentTypeMask:
                             MailSegmentMask; buffer: UNIV Ptr; bufferSize:
                             LONGINT;
                             VAR dataSize: LONGINT;
                             VAR textScrap: StScrpRec;
                             VAR script: ScriptCode;
                             VAR segmentType: MailSegmentType;
                             VAR endOfScript: BOOLEAN;
                             VAR endOfSegment: BOOLEAN;
                             VAR endOfContent: BOOLEAN;
                             VAR segmentLength: LONGINT;
                             VAR segmentID: LONGINT): OSErr;
FUNCTION SMPGetFontNameFromLetter
                            (window: WindowPtr; fontNum: INTEGER; fontName:
                             Str255; doneWithFontTable: BOOLEAN): OSErr;
```

```
FUNCTION SMPGetMainEnclosureFSSpec
                            (window: WindowPtr;
                             VAR enclosureDir: FSSpec): OSErr;
FUNCTION SMPEnumerateBlocks (window: WindowPtr; startIndex: INTEGER;
                             buffer: UNIV Ptr; bufferSize: LONGINT;
                             VAR dataSize: LONGINT; VAR nextIndex: INTEGER;
                             VAR more: BOOLEAN): OSErr;
FUNCTION SMPReadBlock       (window: WindowPtr; blockType: OCECreatorType;
                             blockIndex: INTEGER; buffer: UNIV Ptr;
                             bufferSize: LONGINT; dataOffset: LONGINT;
                             VAR dataSize: LONGINT; VAR endOfBlock:
                             BOOLEAN; VAR remaining: LONGINT): OSErr;
```

Printing Mailers

```
FUNCTION SMPPrepareCoverPages
                            (window: WindowPtr; VAR pageCount: INTEGER):
                             OSErr;
FUNCTION SMPDrawNthCoverPage
                            (window: WindowPtr; pageNumber: INTEGER;
                             doneDrawingCoverPages: BOOLEAN): OSErr;
```

Getting and Setting Information in the Mailer

```
FUNCTION SMPGetComponentSize
                            (window: WindowPtr; whichMailer: INTEGER;
                             whichField: SMPMailerComponent;
                             VAR size: INTEGER): OSErr;
FUNCTION SMPGetComponentInfo
                            (window: WindowPtr; whichMailer: INTEGER;
                             whichField: SMPMailerComponent;
                             buffer: UNIV Ptr): OSErr;
FUNCTION SMPGetListItemInfo (window: WindowPtr; whichMailer: INTEGER;
                             whichField: SMPMailerComponent;
                             buffer: UNIV Ptr; bufferLength: LONGINT;
                             startItem: INTEGER; VAR itemCount: INTEGER;
                             VAR nextItem: INTEGER; VAR more: BOOLEAN):
                             OSErr;
FUNCTION SMPSetSubject      (window: WindowPtr; text: RString): OSErr;
FUNCTION SMPSetFromIdentity
                            (window: WindowPtr; from: AuthIdentity): OSErr;
FUNCTION SMPAddAddress      (window: WindowPtr; addrType: SMPAddressType;
                             address: OCEPackedRecipientPtr): OSErr;
FUNCTION SMPAddAttachment   (window: WindowPtr; attachment: FSSpec): OSErr;
FUNCTION SMPAttachDialog    (window: WindowPtr): OSErr;
```

Application-Defined Functions

```
FUNCTION MyPrepareMailerForDrawing
                            (window: WindowPtr; clientData: LONGINT): void;
FUNCTION MyDrawImage        (refcon: LONGINT; inColor: BOOLEAN): void;
FUNCTION MyFrontWindowCB    (clientData: LONGINT): WindowPtr;
FUNCTION MySendOptionsFilterProc
                            (theDialog: DialogPtr;
                             VAR theEvent: EventRecord; itemHit: INTEGER;
                             clientData: LONGINT): BOOLEAN;
```

Assembly-Language Summary

Trap Macros

Trap Requiring Routine Selectors

$AA5D

Selector	Count	Routine
$01F4	$0002	SMPSendLetter
$044C	$0006	SMPResolveToRecipient
$0834	$0002	SMPNewPage
$0835	$0000	SMPImageErr
$125C	$0006	SMPGetDimensions
$125D	$000C	SMPNewMailer
$125E	$0004	SMPDisposeMailer
$125F	$0008	SMPMailerEvent
$1260	$0005	SMPMailerEditCommand
$1261	$0004	SMPMailerForward
$1262	$000F	SMPMailerReply
$1263	$0004	SMPGetMailerState
$1264	$0004	SMPPrepareCoverPages
$1265	$0004	SMPDrawNthCoverPage
$1266	$000B	SMPBeginSave
$1267	$000A	SMPBeginSend
$1268	$000C	SMPOpenLetter
$1269	$0002	SMPDrawMailer

continued

Selector	Count	Routine
$126A	$0004	SMPMoveMailer
$126B	$0004	SMPSetSubject
$126C	$0004	SMPSetFromIdentity
$126D	$0005	SMPAddAddress
$126E	$0004	SMPAddAttachment
$126F	$0002	SMPContentChanged
$1270	$0002	SMPEndSave
$1271	$0002	SMPEndSend
$1272	$0003	SMPExpandOrContract
$1273	$0005	SMPBecomeTarget
$1274	$0004	SMPGetTabInfo
$1275	$0002	SMPClearUndo
$1276	$0002	SMPAttachDialog
$1277	$0007	SMPGetComponentSize
$1278	$0007	SMPGetComponentInfo
$1279	$0010	SMPGetListItemInfo
$127A	$000D	SMPAddContent
$127B	$0019	SMPReadContent
$127C	$0006	SMPGetFontNameFromLetter
$127D	$0004	SMPAddMainEnclosure
$127E	$0004	SMPGetMainEnclosureFSSpec
$127F	$000C	SMPAddBlock
$1280	$000C	SMPReadBlock
$1281	$000D	SMPEnumerateBlocks
$1282	$0002	SMPImage
$1285	$0002	SMPInitMailer
$1286	$0008	SMPGetNextLetter
$1287	$0002	SMPPrepareToClose
$1288	$0004	SMPCloseOptionsDialog
$1289	$0002	SMPPrepareToChange
$128A	$0004	SMPGetLetterInfo
$128B	$0004	SMPTagDialog
$1388	$0013	SMPSendOptionsDialog

Result Codes

The allocated range of result codes for the Standard Mail Package is –1900 through –1949. Routines may also return standard Macintosh result codes such as `noErr` 0 (no error) and `fnfErr` –43 (file not found).

`kSMPNotEnoughMemoryForAllRecips`	–1900	Too many recipients in mailer
`kSMPCopyInProgress`	–1901	Enclosure being copied to mailer
`kSMPMailerNotInitialized`	–1902	Mailer has not been initialized
`kSMPShouldNotAddContent`	–1903	You cannot add content to letter
`kSMPMailboxNotFound`	–1904	Cannot find mailbox
`kSMPNoNextLetter`	–1905	There is no next letter in In Tray
`kSMPHasOpenAttachments`	–1906	One or more enclosures are open
`kSMPFinderNotRunning`	–1907	The Finder is not running
`kSMPCommandDisabled`	–1908	Requested command unavailable
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPNoSuchAddress`	–1910	Requested address not found
`kSMPMailerAlreadyInWindow`	–1911	A mailer was previously allocated
`kSMPMailerUneditable`	–1912	Mailer cannot be edited
`kSMPNoMatchingBegin`	–1913	End function called without begin
`kSMPCannotSendReceivedLetter`	–1914	Letter is received; cannot be sent
`kSMPIllegalForDraftLetter`	–1915	Operation cannot be completed
`kSMPMailerCannotExpandOrContract`	–1916	Mailer created with `canContract false`
`kSMPMailerAlreadyExpandedOrContracted`	–1917	Mailer is already in requested state
`kSMPIllegalComponent`	–1918	Bad field name parameter
`kSMPMailerAlreadyNotTarget`	–1919	This mailer is not the target
`kSMPComponentIsAlreadyTarget`	–1920	The selected field is the target
`kSMPRecordDoesNotContainAddress`	–1921	Address is not in this record
`kSMPAddressAlreadyInList`	–1922	Specified address is in Recipients field
`kSMPIllegalSendFormats`	–1923	Format is not in `canSend` parameter
`kSMPInvalidAddressString`	–1924	Address string is invalid
`kSMPSubjectTooBig`	–1925	Subject string exceeds 127 characters
`kSMPParamCountErr`	–1926	Enclosure count should be 1
`kSMPTooManyPages`	–1927	Image is more than 127 pages
`kSMPTooManyEnclosures`	–1928	More than 50 total files and folders